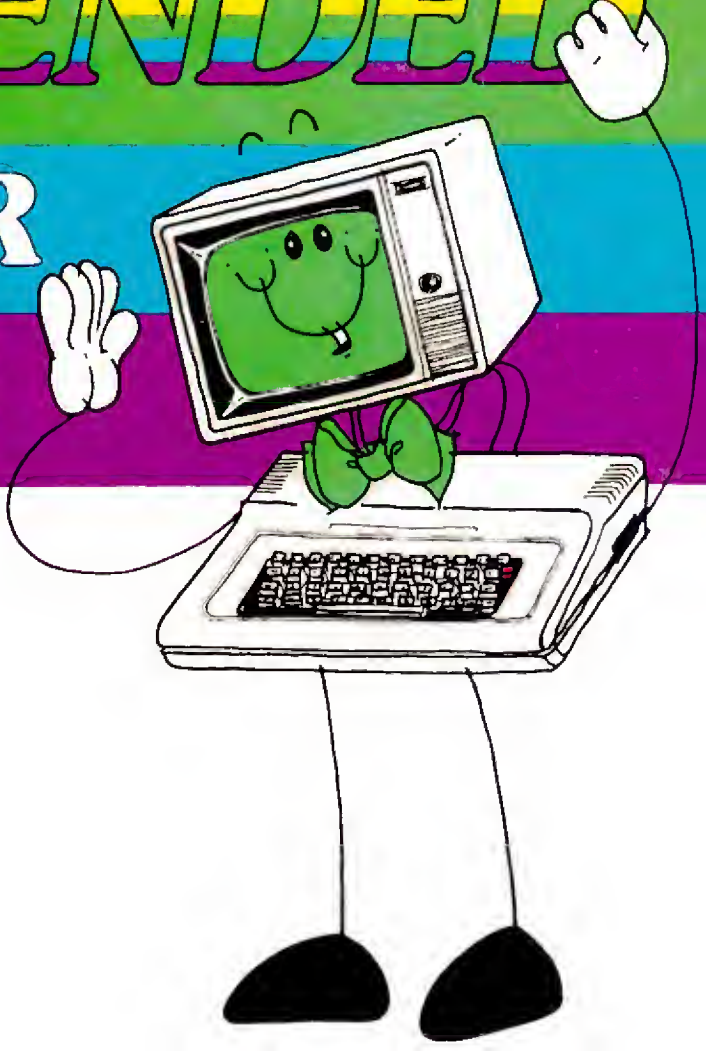
Radio Shack®

GETTING STARTED WITH EXTENDED COLOR BASIC



TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK AND TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment") and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK AND TANDY EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein, no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".
NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

10 9 8 7 6 5 4 3 2 1

To All New Customers . . .

If you don't know a thing about computers, relax — this book's for you! It has you "program" your computer using its own language — Extended Color BASIC. You'll start a little crazy by:

- Composing music
- Playing games
- Conducting light shows
- Painting pictures

If you're straight business, be patient. Having fun's the fastest way to learn.

So spend a few hours with your computer. Type whatever you want. Play with it. Be bold and strange. In other words . . . feel at ease! You have an amazing tool to command.

And to All Upgrading Customers . . .

Welcome back to the Color BASIC family! Let us introduce you to . . . slight drum roll, please . . . Extended Color BASIC. It has *all* the features of non-Extended Color BASIC plus much more.

For example, with Extended Color BASIC you can:

- Draw a circle
- Edit a line
- Paint a house
- Square a root
- Cool off with a cube
- Play a symphony

And even try a triangle!

If you've read *Getting Started with Color BASIC*, you can skip half this book:

- Skip Section I except for Chapter 9. Chapter 9 shows how to use the Extended Color BASIC "Editor" — a great time-saver in typing programs.
- Read Section II. You'll learn to use the most exciting features of Extended Color BASIC — high-resolution graphics and music.
- Skip Section III.
- Read Section IV. This shows how to use the rest of Extended Color BASIC's expanded features.

This Is How to Start . . .

Connect your computer by referring to your *Introducing Your Color Computer 2* or *Introducing Your Deluxe Color Computer*.

Then power up your computer:

1. Turn on your television set
2. Select Channel 3 or 4 on the television set.
3. Set the antenna switch to COMPUTER.
4. Turn on the computer. The POWER button is on the left rear of your keyboard (when you're facing the front).

This message appears on your screen:

```
EXTENDED COLOR BASIC v.r.
© 1980 TANDY
OK
```

(*v.r.* is two numbers specifying which version and release you have.)

If you don't get this message:

- Turn the computer on and off again.
- Adjust the brightness and contrast on your television set.
- Check all the connections.

If you still don't get this message, refer to "Troubleshooting and Maintenance" in *Introducing Your Color Computer 2* or *Introducing Your Deluxe Color Computer*.

Once you do get the above message, you're ready to start.

How Do You Talk to a Computer?

In this book, you'll learn how to talk to your computer. That's all programming is, by the way. Once you learn how to talk to your computer, you can tell it to do whatever you want. (Well, almost.)

Your computer understands a language called Extended Color BASIC. This is an enhanced form of BASIC — Beginners All-Purpose Symbolic Instruction Code. There are lots of computer languages. Extended Color BASIC just happens to be the one your computer understands.

We'll introduce BASIC words in the order that they're easiest to learn. When you get midway in the book, you may forget what a word means. If this happens, simply look it up in your *Quick Reference Card*.

CONTENTS

Section I THE BASICS

Section II SIGHTS AND SOUNDS

Section III GETTING DOWN TO BUSINESS

Section IV BACK TO BASICS

Section V ODDS AND ENDS

SECTION I

THE BASICS

In this section you'll learn how to program. Before you start, though, put yourself in the right frame of mind . . .

Don't try to do everything the "correct" way. Don't try to understand everything. Above all, please don't take our word for anything!

Do have fun with your Color Computer. Try out your own ideas. Prove us wrong (if you can). Type anything and everything that comes to mind.

Ready? Turn the page and begin.

CHAPTER 1

MEET YOUR COMPUTER

Have you connected and turned on your computer? Are you ready to give it a first workout?

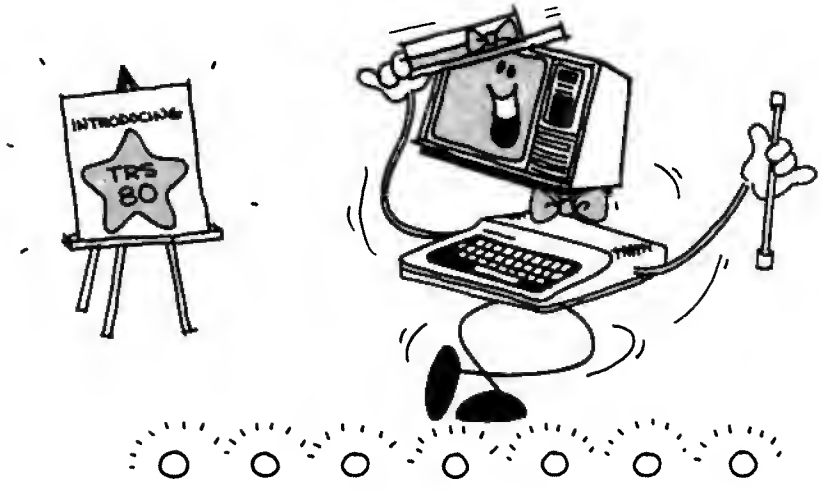
This chapter and the next introduce you to your computer—the way it thinks, some of its talents, and even a couple of its quirks. By the time you reach Chapter 3, you'll be ready to program . . . promise!

Type whatever you want. Then press the (ENTER) key. Don't worry about anything but the last line of type on your screen. It says:

```
OK
```

OK is the computer's "prompt." It's telling you, "OK, enough foolishness . . . as soon as you are ready . . ." (It patiently waits for your command.) You're the master—*you* tell the computer to do whatever you wish.

All letters you type should be BLACK with a GREEN BACKGROUND. If they're reversed (green with a black background), press the (SHIFT) and (0) (zero) keys at the same time.



Give the computer your first command. Type this *exactly* as it is below:

```
PRINT "HI, I'M YOUR COLOR COMPUTER"
```

When you reach the right side of your screen, keep typing. The last part of the message appears on the next line.

Now check your line. Did you put the quotation marks where we have them? If you made a mistake, no problem. Simply press the (←) key and the last character you typed disappears. Press it again and the next to the last disappears (. . . and so on and so on . . .).

See the blinking light? Wherever you see it, you can type something.

Ready? This should be on your screen:

```
OK
PRINT "HI , I'M YOUR COLOR COMPUT
ER"
```

Press the (ENTER) key and watch. Your screen should look like this:

```
OK
PRINT "HI , I'M YOUR COLOR COMPUT
ER"
HI , I'M YOUR COLOR COMPUTER
OK
```

Your computer just obeyed you by printing the message you have in quotes. Have it print another message. Type:

```
PRINT "2"
```

Press (ENTER). The computer again obeys you and prints your next message:

```
2
```

Try another one:

```
PRINT "2 + 2" (ENTER)
```

The computer obeys you by printing:

```
2 + 2
```

You probably expect much more than an electronic mimic . . . maybe some answers! Give your computer some numbers without the quotation marks. Type:

```
PRINT 2 + 2 (ENTER)
```

Much better. This time the computer prints the answer:

```
4
```

The quotation marks obviously have a meaning. Experiment with them some more. Type each of these lines:

```
PRINT 5+4 (ENTER)
PRINT "5+4" (ENTER)
PRINT "5+4 EQUALS" 5+4 (ENTER)
PRINT 6/2 "IS 6/2" (ENTER)
PRINT "8/2" (ENTER)
PRINT 8/2 (ENTER)
```

Any conclusions on what the quotes do?

The computer thinks of quotes as a journalist does. If the number's in quotes, the computer must PRINT it exactly as it appears. If it's not in quotes, the computer can interpret it by adding, subtracting, multiplying, or dividing it.

RULES ON STRINGS *v* NUMBERS

The computer sees everything you type as *strings* or *numbers*. If it's in quotes, it's a *string*. The computer sees it *exactly* as it is. If it's not in quotes, it's a *number*. The computer figures it out like a numerical problem.

A Color Calculator, No Less!

Any arithmetic problem is a snap for the computer. Do some long division. Type:

```
PRINT "3862 DIVIDED BY 13.2 IS" 3862/13.2 (ENTER)
```

Do a multiplication problem:

```
PRINT 1589 * 23 (ENTER)
```

Notice that the computer's multiplication sign is an asterisk (*), rather than the sign you use in math (X). The computer's so precise that it would get the X multiplication sign mixed up with the X alphabetical character.

Try a few more problems:

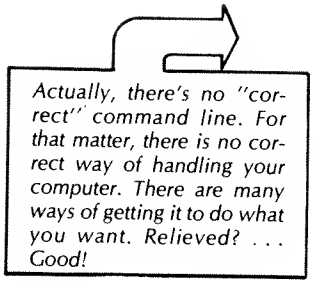
```
PRINT "15 * 2 = " 15*2 (ENTER)
PRINT 18 * 18 "IS THE SQUARE OF 18" (ENTER)
PRINT 33.3/22.82 (ENTER)
```

Notice how the computer handles parts in quotes v parts not in quotes.

Now it's your turn. Write two command lines that print these two problems as well as their answers:

```
157 / 13.2 =
95 * 43 =
```

DO-IT-YOURSELF COMMAND LINES



Actually, there's no "correct" command line. For that matter, there is no correct way of handling your computer. There are many ways of getting it to do what you want. Relieved? . . . Good!

If you use the "correct" command lines, this is what the computer prints on your screen:

```
157 / 13.2 = 11.8939394
95 * 43 = 4085
```

Ready for the answers:

```
PRINT "157 / 13.2 =" 157/13.2
PRINT "95 * 43 =" 95*43
```

It Has Its Rules . . .

By now, the computer has probably printed some funny little messages on your screen. If it hasn't, type this line, deliberately misspelling the word PRINT:

```
PRIINT "HI" (ENTER)
```

The computer prints:

```
?SN ERROR
```

?SN ERROR stands for "syntax" error. This is the computer's way of saying, "The command 'PRIINT' is not in my vocabulary . . . I have no earthly idea what you want me to do." Any time you get the ?SN error, you probably made some kind of typographical mistake.

The computer also gives you error messages when it *does* understand what you want it to do, but it feels you're asking it to do something that is *illogical* or *impossible*. For instance, try this:

```
PRINT 5/0 (ENTER)
```

The computer prints:

```
?/0 ERROR
```

which means, "Don't ask me to divide by 0—that's impossible!"

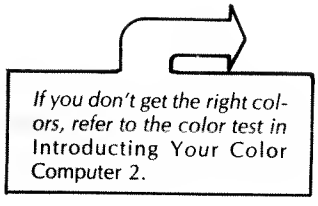
If you get an error message you don't understand, flip to the Appendix. We've listed all the error messages there and what probably caused them.

It's a Show-off Too

So far, all you've seen your computer do is silently print on a green screen. But your color computer enjoys showing off. Type:

```
CLS(3) (ENTER)
```

Now your screen is a pretty shade of blue with a green stripe at the top. Your command told the computer to clear the screen and print color number 3—blue.



If you don't get the right colors, refer to the color test in Introducting Your Color Computer 2.

But why the green stripe? Whenever the computer prints characters, it must use a green background, not a blue background. Type some more characters. The computer uses a green background for them also.

Colors other than green are for printing pictures. You'll learn how to do that later.

Press (ENTER) to get the OK prompt. Then type:

```
CLS(7) (ENTER)
```

Now your screen is magenta (pinkish purple) with a green stripe at the top. Try some more colors. Use any number from 0 to 8. The Color Computer has nine colors. Each color has a numeric code.

BUG: If you see a message saying MICROSOFT, or if you see a ?FC Error message, you're using a number other than 0 through 8.

Type CLS without a number code:

```
CLS (ENTER)
```

If you don't use a number code, the computer assumes you simply want a clear green screen.

Computer Sound Off—One, Two . . .

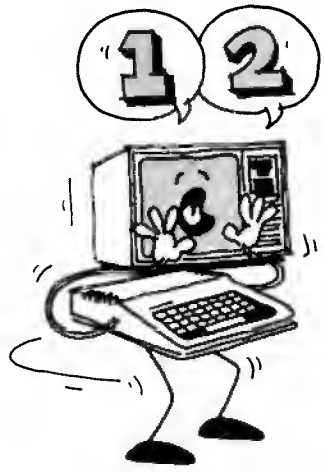
Type this:

```
SOUND 1,100 (ENTER)
```

If you don't hear anything, turn up the volume and try again.

What you're hearing is 6 seconds of the lowest tone the computer can hum. How about the highest tone? Type:

```
SOUND 255,100 (ENTER)
```



OK, so it has a good "hum-range" . . . hope you're suitably impressed. Try some other numbers. Hope you like the computer's voice (it's the only one it has).

You want to know what the other number is for? (Or maybe you've already found out.) The second number tells the computer *how long* to hum the tone. You can use any number from 1 to 255. Try 1:

```
SOUND 128,1 (ENTER)
```

The computer hums the tone for about 6/100ths of a second. Try 10:

BUG: Again, if you get a ?FC Error message, you're using a number other than 1 through 255.

```
SOUND 128,10 (ENTER)
```

The computer sounds the tone for 6/10ths of a second. Try variations of both numbers, but keep in the range of 1 to 255.

Before You Continue . . .

Press the (SHIFT) and (0) (zero) keys, holding both down at the same time. Now release them and type some letters. The letters you type should be *green* on a *black background*. If they're not, try again, pressing (SHIFT) slightly before (0). Be sure to hold down both keys at the same time and then release them.

Curious about the reversed colors? They're for people with a Color Computer 2 and a printer. The printer prints all "reversed" letters in lowercase.

If you have a Deluxe Color Computer, your computer can understand commands in "reversed" or "lowercase' type. See ***Introducing your Deluxe Color Computer*** *to learn how to get in the upper/lower case mode.*

Now, with the colors "reversed," press (ENTER) and then type this simple command line:

```
PRINT "HI" (ENTER)
```

The computer gives you a ?SN ERROR. It doesn't understand the command.

Press the (SHIFT) and (0) characters again and release them. Type some letters. They should be back to normal: *black* with the *green background*. Press (ENTER) and type the same command line again. This time it works.

The computer can't understand any commands you type with reversed colors. If you ever press (SHIFT)(0) by mistake and find you're typing with these reversed colors, press (SHIFT)(0) again to get the colors back to normal.

Learned in Chapter 1

BASIC WORDS	KEYBOARD CHARACTERS	CONCEPTS
PRINT	(←)	string *v* numbers
SOUND	(ENTER)	error messages
CLS		

A refresher like this is at the end of each chapter. It helps you make sure you didn't miss anything.

Notes

CHAPTER 2

YOUR COMPUTER NEVER FORGETS (. . . unless you turn it off . . .)

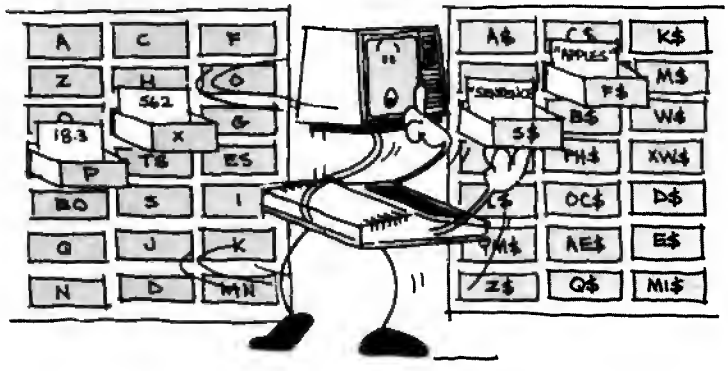
One skill that makes your computer so powerful is its "memory." Have it "remember" the number 13. Type:

```
A = 13 (ENTER)
```

Now "confuse" the computer by typing whatever you want. When you're done, press (ENTER). See if the computer remembers what A means by typing:

```
PRINT A (ENTER)
```

Did it get confused? or forget?



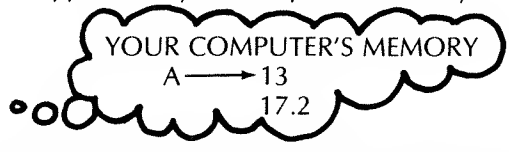
Your computer remembers that A is 13 as long as you have it on . . . or until you do this. Type:

```
A = 17.2 (ENTER)
```

If you ask it to PRINT A now, it prints 17.2.

If you already know BASIC, you may be accustomed to using the word LET before these command lines. The Color Computer doesn't let you use the word LET.

This is what happened in your computer's memory:



You don't have to use the letter A. You can use any letters from A to Z. In fact, you can use any two letters from A to Z. Type:

```
B = 15 (ENTER)
C = 20 (ENTER)
BC = 25 (ENTER)
```

Have it print all the numbers you've asked it to remember. Type:

```
PRINT A, B, C, BC
```

If you want the computer to remember a "string" of letters or numbers, use a letter with a dollar sign ($). Type:

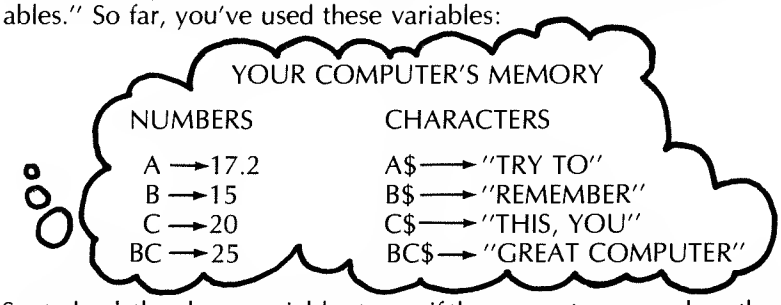
```
A$ = "TRY TO"
B$ = "REMEMBER"
C$ = "THIS, YOU"
BC$ = "GREAT COMPUTER"
```

To the computer, a dollar sign means a string.

Then type:

```
PRINT A$, B$, C$, BC$ (ENTER)
```

"Computer types" have a name for all the letters you've used: "variables." So far, you've used these variables:



Spot-check the above variables to see if the computer remembers the right information. For instance, to see if BC still contains 25, type:

```
PRINT BC (ENTER)
```

Try to set the computer to remember a letter we haven't used yet. What happens? Interesting . . .

Think of variables as little boxes in which you can store information. One set of boxes is for strings; the other set's for numbers. Each box has a label.

As we said before, the computer has its rules and might get a little fussy with you if you don't play by them.

The Computer Is Fussy About Its Rules

Do you think the computer accepts these lines?

```
D = "6" (ENTER)
Z = "THIS IS STRING DATA" (ENTER)
```

TM stands for Type Mis-Match error. It means you didn't go by the rules.

The computer responds to both above lines with ?TM ERROR. It's telling you that you have to play by its rules.

The rules "ignored" by the above lines are:

> RULES ON STRING DATA
>
> (1) Any data in quotes is *STRING DATA*.
>
> (2) You can assign *STRING DATA* only to variables *WITH A $ SIGN*.

To make the above lines obey the computer's rules, use a dollar sign with the D and Z. Type:

```
D$ = "6" (ENTER)
Z$ = "THIS IS STRING DATA" (ENTER)
```

The computer now accepts these lines.

How about this line? Do you think the computer accepts it?

```
D$ = 6 (ENTER)
```

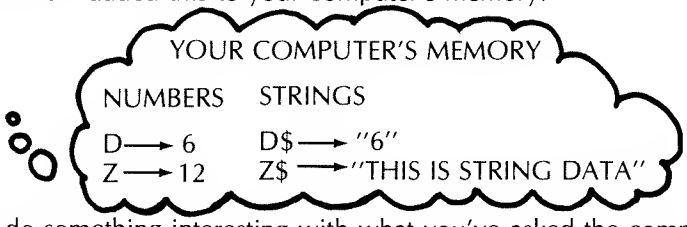
The above line ignored these rules:

> RULES ON NUMERIC DATA
>
> (1) Numbers not in quotes are *NUMERIC DATA*.
>
> (2) Numeric data can only be assigned to variables *WITHOUT A $ SIGN*.

Type this, which the computer accepts:

```
D = 6 (ENTER)
Z = 12 (ENTER)
```

You've now added this to your computer's memory.



Now do something interesting with what you've asked the computer to remember. Type:

```
PRINT D * 2 (ENTER)
```

The computer prints the product of D times 2.

The computer remembers that D = 6.

Try this line:

```
PRINT Z/D
```

The computer prints the quotient of Z divided by D.

Would this work?

```
PRINT D$ * 2 (ENTER)
```

Did you try it? This makes the computer print the same ?TM ERROR. *It cannot multiply string data.*

Cross out the commands below that the computer rejects:

EXERCISE WITH VARIABLES

```
F    = 22.9999999
M    = "19.2"
DZ$ = "REMEMBER THIS FOR ME"
M$   = 15
Z    = F + F
```

Finished? These are the commands the computer *accepts.*

```
F = 22.9999999
DZ$ = "REMEMBER THIS FOR ME"
Z = F + F
```

RULES ON VARIABLES

You may use any two characters from A to Z for a variable. The first character must be a letter from A to Z; however, the second may be either a numeral or a letter. If you want to assign it string data, put a dollar sign after it. Otherwise, it can hold only numeric data.

Learned in Chapter 2

CONCEPTS

Variables
String *v* Numeric Variables

Now that you've learned how the computer thinks, it will be easy to write some programs. How about a break, though, before going to the next chapter?

Notes

SEE HOW EASY IT IS?

Type:

```
NEW (ENTER)
```

This erases whatever may be in the computer's "memory."

Now type this line. Be sure you type the number 10 first—that's important.

```
10  PRINT "HI, I'M YOUR COLOR COMPUTER" (ENTER)
```

Did you press (ENTER)? Nothing happened, did it? Nothing you can see, that is. You just typed your first program. Type:

```
RUN (ENTER)
```

The computer obediently runs your program. Type RUN again and again to your heart's content. The computer runs your program any time you wish, as many times as you wish.

Since this works so well, add another line to the program. Type:

```
20  PRINT "WHAT IS YOUR NAME?" (ENTER)
```

Now type:

```
LIST (ENTER)
```

Your computer obediently *lists* your entire program. Your screen should look exactly like this:

```
10  PRINT "HI, I'M YOUR COLOR COM
PUTER"
20  PRINT "WHAT IS YOUR NAME?"
```

What do you think will happen when you run this? Try it. Type:

```
RUN (ENTER)
```

The computer prints:

```
HI, I'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
```

Answer the computer's question and then press (ENTER). . . . What? There's the ?SN Error again.

When you simply type your name, the computer doesn't understand what you mean. In fact, *the computer can't understand anything unless you talk to it in its own way.*

If you make a mistake after pressing (ENTER), simply type the line over again.

Use a word the computer understands: INPUT. Type this line:

```
30  INPUT A$ (ENTER)
```

This tells the computer to stop and wait for you to type something, which it labels as A$. Add one more line to the program:

```
40  PRINT "HI ," A$ (ENTER)
```

Now list the program again to see if yours looks like mine. Type:

```
LIST (ENTER)
```

Your program should look like this:

```
10  PRINT "HI , I 'M YOUR COLOR COM
PUTER"
20  PRINT "WHAT IS YOUR NAME?"
30  INPUT A$
40  PRINT "HI ," A$
```

Can you guess what will happen when you run it? Try it:

```
RUN (ENTER)
```

That worked well, didn't it? This is probably what happened when you ran the program (depending on what you typed as your name):

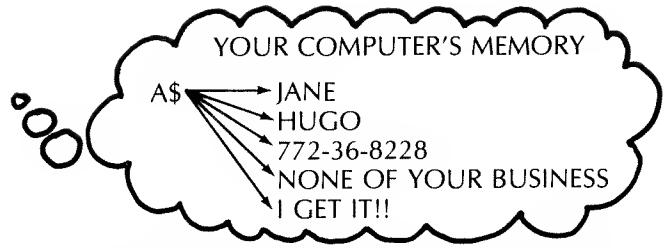
```
HI , I 'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
? JANE
HI , JANE
```

RUN the program again using different names:

```
HI , I 'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
? HUGO
HI , HUGO

HI , I 'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
? 772-36-8228
HI , 722-36-8228

HI , I 'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
? NONE OF YOUR BUSINESS
HI , NONE OF YOUR BUSINESS

HI , I 'M YOUR COLOR COMPUTER
WHAT IS YOUR NAME?
? I GET IT!!
HI , I GET IT!!
```

(The computer doesn't care what you call yourself.)

Here's what Line 30 did to your computer's memory each time you ran the program (assuming you gave it the same names we did):



There's an easier way to run your program over and over without having to type the RUN command. Type this line:

```
50  GOTO 10
```

Now run it. The program runs over and over again without stopping. GOTO tells the computer to go back to Line 10:

```
10  PRINT "HI, I'M YOUR COLOR COMPUTER"
20  PRINT "WHAT IS YOUR NAME?"
30  INPUT A$
40  PRINT "HI," A$
50  GOTO 10
```

Your program now runs perpetually. Each time it gets to Line 50, it goes back to Line 10. We call this a "loop." The only way you can stop this endless loop is by pressing the (BREAK) key.

Spotlight Your Name

To delete a program line, simply type and (ENTER) the line number. For example:
50 (ENTER)
erases line 50 from the program.

Change Line 50 to give your name the attention it deserves. How do you change a program line? Simply type it again, using the same line number. Type:

```
50  GOTO 40
```

This is what the program looks like now:

```
10  PRINT "HI, I'M YOUR COLOR COMPUTER"
20  PRINT "WHAT IS YOUR NAME?"
30  INPUT A$
40  PRINT "HI," A$
50  GOTO 40
```

Type RUN and watch what this loop does. When you've seen enough, press the (BREAK) key.

There's a big change you can make simply by adding a comma or a semicolon. Try the comma first. Type Line 40 again, but with a comma at the end:

We're leaving out the "HI," part this time.

```
40  PRINT A$,
```

Run the program. The comma seems to print everything in two columns.

Press (BREAK) and try the semicolon. Type:

```
40  PRINT A$;
```

and run . . . You probably won't be able to tell what the program's doing until you press (BREAK). See how the semicolon crams everything together?

Remember, if you make a mistake on one of the lines, simply type the line over again.

RULES ON PRINT PUNCTUATION

This is what punctuation at the end of a PRINT line makes the computer do:

1. A *comma* makes the computer go to the next column. Use it to print in columns.
2. A *semicolon* makes the computer stay where it is. Use it to "cram" what you print together.
3. *No punctuation* makes the computer go to the next line. Use it to print in rows.

Color/Sound Demonstration

NEW (ENTER) . . . wish mine worked that easily!

Want to play with color and sound some more? First, erase memory. Remember how?

Then enter this program:

```
10  PRINT "TO MAKE ME CHANGE MY TONE"
20  PRINT "TYPE IN A NUMBER FROM 1 TO 255"
30  INPUT T
40  SOUND T, 50
50  GOTO 10
```

In this program we're using T as a variable. However, we could use any letter.

Notice that line 30 asks for T rather than T$. This is because we want numeric data rather than string data.

Run through the program to get a sample of the computer's tones.

BUG: If you get a ?FC Error when you run this program, you used a number other than 1 through 255. This error, like all errors, will make the computer stop running the program.

What happens if you change Line 40 to:

```
40  SOUND 50, T
```

HINT: Look back in Chapter 1 where we talk about SOUND.

Know the answer? If you make the above change, the computer hums the same tone each time, but for a different length of time, depending on what number you use.

DO-IT-YOURSELF PROGRAM

Press (BREAK) first and then erase this program by typing NEW. Now see if you can write a program, similar to the one above, to make the computer show a certain color. Remember, there are 9 colors, 0 through 8.

HINT: Line 40 could be: 40 CLS(T).

This is our program:

```
10  PRINT "TO MAKE ME CHANGE MY COLOR"
20  PRINT "TYPE A NUMBER BETWEEN Ø AND 8"
30  INPUT T
40  CLS(T)
50  GOTO 10
```

Add Polish to the Program

Press (BREAK) before typing the line.

Pressing the (BREAK) key is a sloppy way to stop the program from running. Why not have the computer politely ask if you're ready to end? Change Line 50 in the above program to:

```
50  PRINT "DO YOU WANT TO SEE ANOTHER COLOR?"
```

Then add these lines:

```
60  INPUT R$
70  IF R$ = "YES" THEN 20
```

Run the program. Type YES and the program keeps running. Type anything else and the program ends.

This is what the program looks like now:

```
10  PRINT "TO MAKE ME CHANGE COLORS"
20  PRINT "TYPE A NUMBER BETWEEN Ø AND 8"
30  INPUT T
40  CLS(T)
50  PRINT "DO YOU WANT TO SEE ANOTHER COLOR"
60  INPUT R$
70  IF R$ = "YES" THEN 20
```

This is what the new lines do:

Don't worry about IF/THEN right now. We devote a whole chapter to it later.

- Line 50 prints a question.
- Line 60 tells the computer to stop and wait for an answer: R$.
- Line 70 tells the computer to go back to Line 20 *IF* (and only if) your answer (R$) is "yes." If not, the program ends, since it has no more lines.

You've covered a lot of ground in this chapter. Hope we're just whetting your appetite for more.

Don't worry if you don't yet understand it perfectly. Just enjoy using your computer.

Learned in Chapter 3

BASIC WORDS	CONCEPT	KEYBOARD
Characters NEW INPUT GOTO RUN PRINT, PRINT; LIST IF/THEN	How to Change and Delete a Program Line	(BREAK)

Notes

COUNT THE BEAT

The logic of this will become clear later.

In this chapter you'll experiment with computer sound effects. Before doing this, you need to teach the computer to count.

Type:

```
10  FOR X = 1 TO 10
20  PRINT "X = " X
30  NEXT X
40  PRINT "I HAVE FINISHED COUNTING"
```

Remember to type NEW ENTER before typing a new program.

Run the program.

Run the program a few more times. Each time, replace Line 10 with one of these lines:

```
10  FOR X = 1 TO 100
10  FOR X = 5 TO 15
10  FOR X = -2 TO 2
10  FOR X = 20 TO 24
```

Do you see what FOR and NEXT make the computer do? They make it count. Look at the last program we suggested you try:

```
10  FOR X = 20 TO 24
20  PRINT "X = " X
30  NEXT X
40  PRINT "I HAVE FINISHED COUNTING"
```

Line 10 tells the computer the first number should be 20 and the last number should be 24. It uses X to label all these numbers.

Line 30 tells the computer to keep going back to Line 10 for the next number—the NEXT X—until it reaches the last number (number 24).

Look at Line 20. Since Line 20 is between the FOR and NEXT lines, the computer must print the value of X each time it counts:

```
X = 20
X = 21
X = 22
X = 23
X = 24
```

Add another line between FOR and NEXT:

```
15  PRINT "... COUNTING ..."
```

and run the program. With each count, your computer runs any lines you choose to insert between FOR and NEXT.

DO-IT-YOURSELF PROGRAM 4-1

Write a program that makes the computer print your name 10 times.

HINT: The program must count to 10.

DO-IT-YOURSELF PROGRAM 4-2

Write a program to print the multiplication tables for 9 (9*1 through 9*10).

HINT: PRINT 9*X is a perfectly legitimate program line.

DO-IT-YOURSELF PROGRAM 4-3

Write a program that prints the multiplication tables for 9*1 through 9*25.

HINT: By adding a comma in the PRINT line, you can get all the problems and results on your screen at once.

Finished? These are our programs:

Program 4-1

```
10 FOR X = 1 TO 10
20 PRINT "THOMAS"
30  NEXT X
```

Program 4-2

```
10 FOR X = 1 TO 10
20 PRINT "9*"X"="9*X
30 NEXT X
```

Program 4-3

```
10 FOR X = 1 TO 25
20 PRINT "9*"X"="9*X,
30 NEXT X
```

Counting by Twos

Now make the computer count somewhat differently. Erase your program by typing NEW and then type the original program, using a new Line 10:

```
10  FOR X = 2 TO 10 STEP 2
20  PRINT "X= " X
30  NEXT X
40  PRINT "I HAVE FINISHED COUNTING"
```

Run the program. Do you see what the STEP 2 does? It makes the computer count by 2s. Line 10 tells the computer that:

The first X is 2

The last X is 10

. . . AND STEP 2 . . .

All the Xs between 2 and 10 are two apart . . . that is 2, 4, 6, 8, and 10. (STEP 2 tells the computer to add two to get each NEXT X.)

To make the computer count by 3s, make all the Xs three apart. Try this for Line 10:

```
10  FOR X = 3 TO 10 STEP 3
```

Run the program. This prints on your screen:

```
X = 3
X = 6
X = 9
```

It passes up the last X (number 10) because 9 + 3 = 12. Try a few more FOR . . . STEP lines so you can see more clearly how this works:

```
10  FOR X = 5 TO 50 STEP 5
10  FOR X = 10 TO 1 STEP-1
10  FOR X = 1 TO 20 STEP 4
```

You may be wondering about the programs you ran at the first of this chapter without using STEP. If you omit STEP, the computer assumes you mean STEP 1.

Counting the Sounds

Now that you've taught the computer to count, you can add some sound. Erase your old program and type this:

```
10  FOR X = 1 TO 255
20  PRINT "TONE " X
30  SOUND X, 1
40  NEXT X
```

Don't type the arrow, of course. It's there to help you understand.

This program makes the computer count from 1 to 255 (by 1s). Each time it counts a new number, it does what Lines 20 and 30 tell it to do:

- Line 20—It prints X, the current count.
- Line 30—It sounds X's tone.

For example:

- The first time the computer gets to FOR, in Line 10, it makes X equal to 1.
- Then it goes to Line 20 and prints 1, the value of X.
- Then Line 30 has it sound tone #1.
- Then it goes back to Line 10 and makes X equal to 2
- Etc.

What do you think the computer will do if you make this change to Line 10:

```
10  FOR X = 255 TO 1 STEP -1
```

Did you try it?

PROGRAMMING EXERCISE

Using STEP, change Line 10 so the computer will sound tones from:

(1) The bottom of its range to the top, humming every tenth note.

(2) The top of its range to the bottom, humming every tenth note.

(3) The middle of its range to the top, humming every fifth note.

10_____

10_____

10_____

Ready for the answers?

```
10  FOR X = 1 TO 255 STEP 10
10  FOR X = 255 TO 1 STEP -10
10  FOR X = 128 TO 255 STEP 5
```

Try this: To pause the program while it's running, press the SHIFT and @ keys at the same time. Then press any key to continue.

DO-IT-YOURSELF PROGRAM 4-4

Now see if you can write a program that makes the computer hum:

(1) from the bottom of its range to the top, and then
(2) from the top of its range back to the bottom

The answer is in the back of this book.

But Can It Sing?

Yes. In Section II, you'll learn how to compose your favorite songs.

Learned in Chapter 4

BASIC WORDS	KEYBOARD CHARACTER
FOR . . . TO . . . STEP NEXT	SHIFT @

Notes

CHAPTER 5

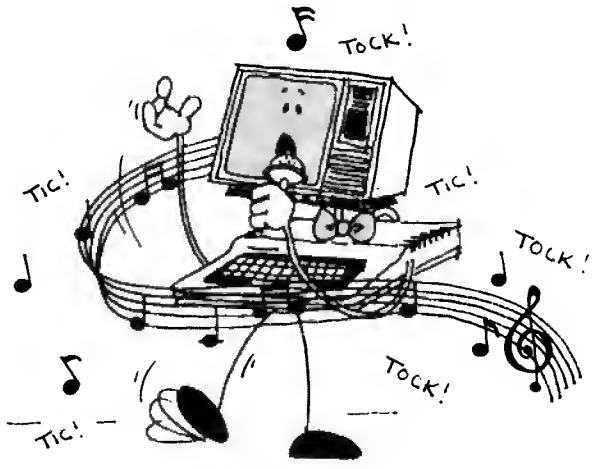
Watch the Clock

You're now ready to show your computer how to tell time. Type:

```
10  FOR Z = 1 TO 460 * Z
20  NEXT Z
30  PRINT "I COUNTED TO 920"
```

Run the program. Be patient and wait a couple of seconds. Two seconds, to be precise. It takes your computer two seconds to count to 920.

Lines 10 and 20 set a *timer pause* in your program. By making the computer count to 920, you keep the computer busy for two seconds.



As you can see, this is groundwork for a stopwatch. Erase the program and type:

```
10  PRINT "HOW MANY SECONDS?"
20  INPUT S
30  FOR Z = 1 TO 460*S
40  NEXT Z
50  PRINT S " SECONDS ARE UP!!!"
```

Run it. Input the number of seconds you want timed on your stopwatch.

DO-IT-YOURSELF PROGRAM 5-1

It would help if the stopwatch could sound some kind of alarm. Add lines to the end of the program to give it an alarm.

Here's the program we wrote:

```
10  PRINT "HOW MANY SECONDS"
20  INPUT S

30  FOR Z = 1 TO 460 * S
40  NEXT Z

50  PRINT S " SECONDS ARE UP!!!"

60  FOR T = 120 TO 180
70  SOUND T, 1
80  NEXT T

90  FOR T = 150 TO 140 STEP -1
100  SOUND T, 1
110  NEXT T

120  GOTO 50
```

This is how computerized timers work.

Notice the GOTO line at the end of the program. It causes the message to keep printing and the alarm to keep ringing until you press BREAK or SHIFT @.

Counting Within the Time

Before doing more with the clock, have the computer keep count *within* the time. This concept will become clear to you shortly.

Type this new program:

```
10  FOR X = 1 TO 3
20  PRINT "X = " X
30  FOR Y = 1 TO 2
40 PRINT, "Y=" Y
50  NEXT Y
60  NEXT X
```

Notice the comma in Line 40. Try it without the comma. The comma makes "Y = " Y print on the next column.

Run it. This should be on your screen:

```
X = 1
                    Y = 1
                    Y = 2
X = 2
                    Y = 1
                    Y = 2
X = 3
                    Y = 1
                    Y = 2
```

Call it a count within a count or a loop within a loop—whatever you prefer. Programmers call this a "nested loop." This is what the program does:

I. It counts X from 1 to 3. *Each time* it counts X:

   A. It prints the value of X

   B. It counts Y from 1 to 2. *Each time* it counts Y:

      1. It prints the value of Y

Whenever you put a loop inside another loop, you must close the inner loop before closing the outer loop:

Right

```
10  FOR X = 1 TO 3
20  FOR Y = 1 TO 2
30  NEXT Y
40  NEXT X
```

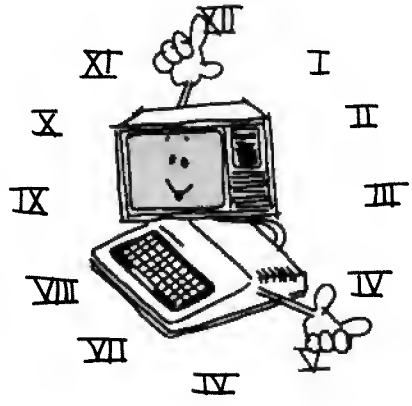
Wrong

```
10  FOR X = 1 TO 3
20  FOR Y = 1 TO 2
30  NEXT X
40  NEXT Y
```

Making a Clock

With these tools, you can make the computer do much more. Type this:

```
10  FOR S = 0 TO 59
20  PRINT S
30  SOUND 150, 2
40  FOR T = 1 TO 390
50  NEXT T
60  NEXT S
70  PRINT "1 MINUTE IS UP"
```



Run the program. This is what it does:

I. It counts the seconds from 0 to 59. *Each time* it counts one second:

   A. It prints the second.

   B. It sounds a tone.

   C. It pauses long enough for one second to pass.

II. When it finishes counting all the seconds from 0 to 59, it prints a message that one minute is up.

There's a way to make this program look better. Add this line to clear the screen:

```
15 CLS
```

Now run the program. This time the computer goes through these steps:

I. It counts the seconds from 0 to 59 (Lines 10 and 60). *Each time* it counts one second:

   A. It clears the screen (Line 15).

   B. It prints the second (Line 20).

   C. It sounds a tone (Line 30).

   D. It pauses long enough for one second to pass (Lines 40 and 50).

II. When it finishes counting all the seconds from 0 to 59, it prints a message that one minute has passed (Line 70).

Using this as groundwork, it's easy to make a full-fledged clock:

```
10  FOR H = 0 TO 23
20  FOR M = 0 TO 59
30  FOR S = 0 TO 59
40  CLS
50  PRINT H";":"M";":"S
60  SOUND 150, 2
70  FOR T = 1 TO 375
80  NEXT T
90  NEXT S
100  NEXT M
110  NEXT H
```

Here's an outline of what the computer does in this program:

I. It counts the hours from 0 to 23 (Line 10). *Each time* it counts a new hour:

   A. It counts the minutes from 0 to 59 (Line 20). *Each time* it counts a new minute:

      1. It counts the seconds from 0 to 59 (Lines 30 and 90). *Each time* it counts a new second:

         a. It clears the screen (Line 40).
         b. It prints the hour, minute, and second (Line 50).
         c. It sounds a tone (Line 60).
         d. It pauses long enough for one second to pass (Lines 70 and 80).

      2. When it finishes counting all the 59 seconds, it goes back to Line 20 for the next minute (Line 100).

   B. When it finishes counting all the 59 minutes, it goes back to Line 10 for the next hour (Line 110).

II. When it finishes counting all the hours (0-23), the program ends.

By adding this line, 120 GOTO 10, the clock will run perpetually.

Having a tough time with this program? Skip it for now. It'll seem easy later.

DO-IT-YOURSELF PROGRAM 5-2

Between Lines 90 and 100 you can add some tones that will sound each minute. Write a program that does this.

DO-IT-YOURSELF PROGRAM 5-3

Write a program that makes your computer show each of its nine colors for 1 second each.

The answers to both programs are in the back.

Learned in Chapter 5

BASIC WORD	PROGRAMMING CONCEPT
CLS	Nested Loops

Notes

DECISIONS, DECISIONS . . .

Here's an easy decision for the computer:

If you type "red" . . . *then* make the screen red

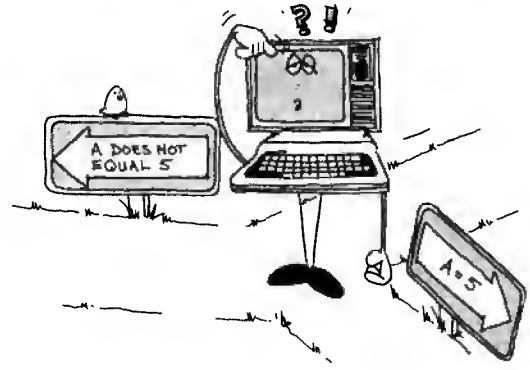
. . . or

If you type "blue" . . . *then* make the screen blue

Easy enough? Then have the computer do it. Type this program:

```
10  PRINT "DO YOU WANT THE SCREEN RED OR BLUE?"
20  INPUT C$
30  IF C$ = "RED" THEN 100
40  IF C$ = "BLUE" THEN 200
100  CLS(4)
110  END
200  CLS(3)
```

Don't be confused by the arrows or the spaces between program lines. We just put them in to illustrate the flow of the program.



Run the program a few times. Try both "red" and "blue" as answers.

This is what the program does:

If you answer "red" . . . *then* . . .

1. Line 30 sends the computer to Line 100.
2. Line 100 turns your screen red.
3. Line 110 ends the program. (If the computer gets to Line 110, it never makes it to 200.)

. . . On the other hand . . .

If you answer "blue" . . . *then* . . .

1. Line 40 sends the computer to Line 200.
2. Line 200 turns your screen blue.
3. Since Line 200 is the last line in the program, the program ends there.

What happens if you answer with something different from "red" or "blue"? Run the program again. This time, answer "green."

This makes the screen red. Do you know why?

HINT: If the condition is not true, the computer ignores the THEN part of the line and proceeds to the next program line.

PROGRAMMING EXERCISE

There's a way to get this program to reject any answer but "red" or "blue." These are the two lines to add. You figure out where they go in the program:

.... PRINT "YOU MUST TYPE EITHER RED OR BLUE"

.... GOTO 20

Insert the line numbers.

HINT: The lines must come *after* the computer has had a chance to test your answer for "red" or "blue."

HINT: The lines must come *before* the computer makes your screen "red."

Answer: The lines need to come after Line 40 and before Line 100:

```
50  PRINT "YOU MUST TYPE EITHER RED OR BLUE"
60  GOTO 20
```

DO-IT-YOURSELF PROGRAM 6-1

After the computer turns the screen red or blue, have it go back and ask you to type "red" or "blue" again.

HINT: You need to change Line 110 and add Line 210.

Here's a diagram of how we wrote this program.

```
10   PRINT "DO YOU WANT THE SCREEN RED OR BLUE?"
20   INPUT C$

30   IF C$ = "RED" THEN 100

40   IF C$ = "BLUE" THEN 200

50   PRINT "YOU MUST TYPE EITHER RED OR BLUE"

60   GOTO 20

100   CLS(4)
110   GOTO 10

200   CLS(3)
210   GOTO 10
```

Trace the path the computer takes through this program. Go from one line to the next; follow the arrows where indicated. Notice the difference between the arrows going from the IF/THEN and the GOTO lines.

RULES ON IF/THEN AND GOTO

IF/THEN is conditional. The computer "branches" only if the condition is true.

GOTO is unconditional. The computer always branches.

Although this chapter is short, you've learned an important programming concept. You'll have the computer make decisions all through this book.

Learned in Chapter 6

BASIC WORDS

IF/THEN
END

Notes

CHAPTER 7

GAMES OF CHANCE

Thanks to a BASIC word called RND, the computer can play almost any game of chance.

And even if you don't want to play computer games, you'll want to learn two words this chapter introduces: RND and PRINT @. You'll also find in this chapter some more uses of IF/THEN.

Type this program:

```
10  PRINT RND(10)
```

Run it. The computer just picked a random number from 1 to 10. Run it some more times . . .

It's as if the computer is drawing a number from 1 to 10 out of a hat. The number it picks is unpredictable.

To make the computer pause while running the program, press the (SHIFT) *and* (@) *keys at the same time. Press any key to continue.*

Type and run this next program. Press (BREAK) when you satisfy yourself that the numbers are random.

```
10  PRINT RND(10);
20  GOTO 10
```

To get random numbers from 1 to 100, change Line 10 and run the program.

```
10  PRINT RND(100);
```

How can you change the program to get random numbers from 1 to 255?

.....

The answer is:

```
10  PRINT RND(255);
```

A Random Show

Just for fun, have the computer compose a song made up of random tones. Type:

```
10  T = RND(255)
20  SOUND T, 1
30  GOTO 10
```

Run it. Great music, eh? Press (BREAK) when you've heard enough.

Sneak preview: Enjoying graphics and sound? Go ahead and try out some programs in Section II, "Sights and Sounds."

DO-IT-YOURSELF PROGRAM 7-1

Add some lines to make the computer show a random color (1-8) just before it sounds each random tone.

Here's our program:

```
10  T = RND(255)
14  C = RND(8)
16  CLS(C)
20  SOUND T, 1
30  GOTO 10
```

We have a few simple games in this chapter. Feel free to use your imagination to add interest to them—or invent your own.

Russian Roulette

In this game, a gun has 10 chambers. The computer picks, at random, which of the 10 chambers has the fatal bullet. Type:

Remember always to type NEW (ENTER) before entering a new program.

```
10  PRINT "CHOOSE YOUR CHAMBER(1-10)"
20  INPUT X
30  IF X = RND(10) THEN 100
40  SOUND 200, 1
50  PRINT "--CLICK--"
60  GOTO 10

100  PRINT "BANG--YOU'RE DEAD"
```

First, in Line 20, the player inputs X (a number from 1 to 10). Then, the computer compares X with RND(10) (a random number from 1 to 10).

Then it follows the "arrows":

If X is equal to RND(10), the computer goes to Line 100, the "dead routine."

If X is not equal to RND(10), the computer "clicks" and goes back to Line 10, where you get another chance . . .

Remember how to list part of a program? LIST 50-130 lists the program's middle part.

Try this when listing a long program: At the start of the listing, press (SHIFT) and (@). This causes the listing to pause. Then press any key to continue.

Make the dead routine in Line 100 better. Type:

```
100 FOR T = 133 TO 1 STEP -5
110 PRINT "              BANG!!!!!"
120 SOUND T, 1
130 NEXT T
140 CLS
150 PRINT @ 230, "SORRY, YOU'RE DEAD"
160 SOUND 1, 50
170 PRINT @ 390, "NEXT VICTIM, PLEASE"
```

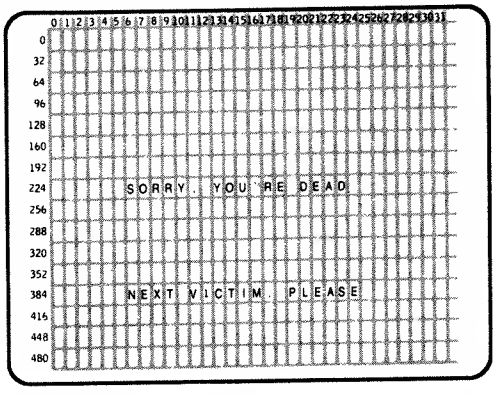
Run the program. Here's what the routine does:

Lines 100-130 make the computer sound descending tones and print BANG!!!!! over and over again on the screen.

Line 140 clears the screen. Since no color is given, the computer makes the screen green.

Lines 150 and 170 use a new word—PRINT @—to position two messages on your screen: SORRY, YOU'RE DEAD and NEXT VICTIM, PLEASE.

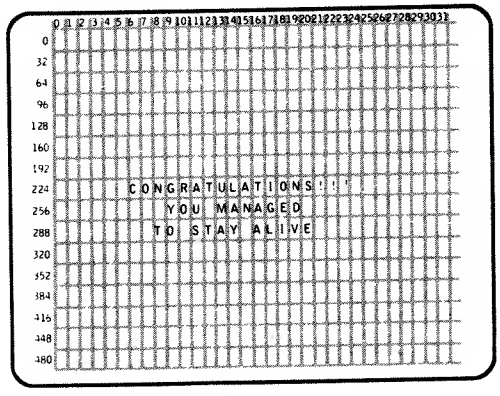
The grid below shows the 511 positions on your screen. Line 150 prints SORRY, YOU'RE DEAD at position 230 (224 + 6). Line 170 prints NEXT VICTIM, PLEASE at position 390 (384 + 6).



The grid is in the Appendix, "PRINT @ Screen Locations." Use it to plan your programs' screen formats.

DO-IT-YOURSELF PROGRAM 7-2

Change this program so that if the player *does* manage to stay alive for 10 clicks, the computer pronounces the player the winner, printing this message on the screen:



HINT: You can use the FOR/NEXT loop, so that the computer can keep count of the number of clicks.

Our answer is in the Appendix.

Rolling the Dice

This game has the computer roll two dice. To do this, it must come up with two random numbers. Type:

```
10  CLS
20  X = RND(6)
30  Y = RND(6)
40  R = X + Y
50  PRINT @ 200, X
60  PRINT @ 214, Y
70  PRINT @ 394, "YOU ROLLED A" R
80  PRINT @ 454, "DO YOU WANT ANOTHER ROLL?"
90  INPUT A$
100  IF A$ = "YES" THEN 10
```

Run the program.

Line 10 clears the screen.

Line 20 picks a random number from 1 to 6 for one die. Line 30 picks a random number for the other die.

Line 40 adds the two dice to get the total roll.

Lines 50-70 print the results of the roll.

Line 90 lets you input whether you want another roll. If you answer "yes," the program goes to Line 10 and runs again. Otherwise, since this is the last line in the program, the program ends.

DO-IT-YOURSELF PROGRAM 7-3

Since you know how to roll dice, it should be easy to write a "Craps" program. These are the rules of the game (in its simplest form):

1. The player rolls two dice. If the first roll's a 2 ("snake eyes"), a 3 ("cock-eyes"), or a 12 ("boxcars"), the player loses and the game's over.
2. If the first roll's a 7 or 11 ("a natural"), the player wins and the game's over.
3. If the first roll's any other number, it becomes the player's "point." The player must keep rolling until either "making the point" by getting the same number again to win, or rolling a 7, and losing.

You already know more than enough to write this program. Do it. Make the computer print it in an attractive format on your screen and keep the player informed about what is happening. It may take you a while to finish, but give it your best. Good luck!

Our answer's in the back.

Learned in Chapter 7

BASIC WORDS

RND
PRINT @

Notes

CHAPTER 8

READING

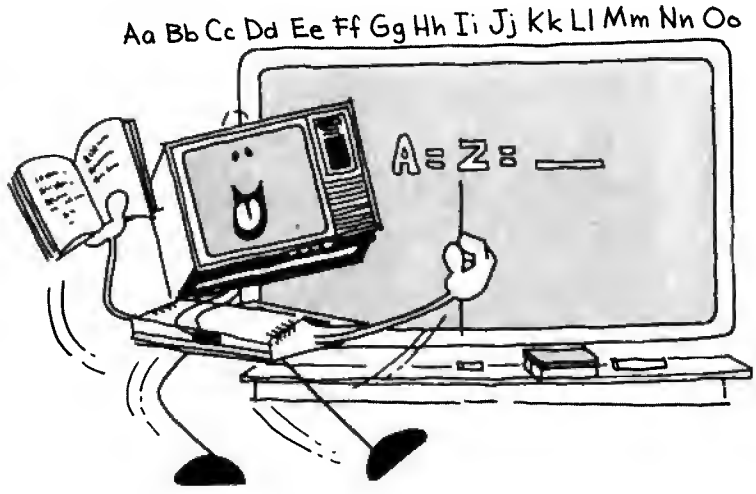
Your computer is a natural at teaching. It's patient, tireless, and never makes a mistake. Depending on the programmer (*you*, of course), it also can be imaginative, consoling, and enthusiastic.

Using RND, have it teach you math. Type:

```
10  CLS
20  X = RND(15)
30  Y = RND(15)
40  PRINT "WHAT IS" X "*" Y " ? "
45  INPUT A
50  IF A = X * Y THEN 90
60  PRINT "THE ANSWER IS" X*Y
70  PRINT "BETTER LUCK NEXT TIME"
80  GOTO 100
90  PRINT "CORRECT!!!"
100   PRINT "PRESS <ENTER> WHEN READY FOR
      ANOTHER"
105   INPUT A$
110   GOTO 10
```

The above program drills you on the multiplication tables, from 1 to 15, and checks your answers.

Are your programs getting long? If you have a cassette recorder, read your computer's introduction manual to learn how to save your programs on tape. If you have a Deluxe Color Computer, you can also save programs in memory. See your introduction manual to learn how.



DO-IT-YOURSELF PROGRAM 8-1

Make the program drill you on addition problems from 1 to 100.

Here are the lines we changed:

```
20  X = RND(100)
30  Y = RND(100)
40  PRINT "WHAT IS" X "+" Y
45  INPUT A
50  IF A = X + Y THEN 90
60  PRINT "THE ANSWER IS" X + Y
```

Make the program more interesting. Have it keep a running total of all the correct answers. Type:

```
15  T = T + 1
95  C = C + 1
98  PRINT "THAT IS" C "CORRECT OUT OF" T
    "ANSWERS"
```

When you first turn on the computer, all numeric variables equal 0. When you type NEW (ENTER), all numeric variables also equal 0.

T is a "counter." It counts how many questions you're asked. When you first start the program, T equals zero. Then each time the computer gets to Line 15, it adds 1 to T.

C is also a counter. It counts your correct answers. Since C's in Line 95, the computer doesn't increase C unless your answer's correct.

DO-IT-YOURSELF PROGRAM 8-2

Make the program more fun. Have it do one or more of the following:

1. Call you by name.
2. Reward your correct answer with a sound and light show.
3. Print the problem and messages attractively on your screen. (Use PRINT @ for this.)
4. Keep a running total of the percentage of correct answers.
5. End the program if you get 10 answers in a row correct.

Use your imagination. We have a program in back that does this all.

First, Build Your Computer's Vocabulary . . .

To build your computer's vocabulary (so that it can build yours!), type and run this program:

```
10  DATA APPLES, ORANGES, PEARS
20  FOR X = 1 TO 3
30  READ F$
40  NEXT X
```

What happened . . . nothing? Nothing that you can see, that is. To see what the computer is doing, add this line and run the program:

```
35  PRINT "F$ = :" F$
```

Line 30 tells the computer to:

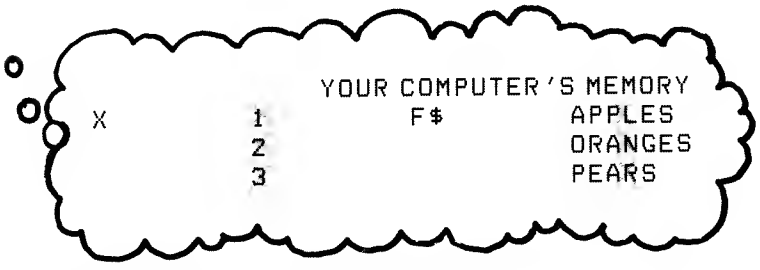
1. Look for a DATA line.
2. READ the first item in the list—APPLES.

3. Give APPLES an F$ label.
4. "Cross out" APPLES.

The second time the computer gets to Line 30 it is told to do the same:

1. Look for a DATA line.
2. READ the first item—this time, it's ORANGES.
3. Give ORANGES the F$ label.
4. "Cross out" ORANGES.

When you run the program, this happens in the computer's memory:



What if you want the computer to read the same list again? It's already "crossed out" all the data . . . Type:

```
60  GOTO 10
```

Run the program. You get an error: ?OD ERROR IN 30. OD means "out of data." The computer's crossed out all the data.

Type this line and run the program:

```
50  RESTORE
```

Now it's as if the computer never crossed out any data. It reads the same list again and again.

Remember how to make the computer pause while running a program? Press (SHIFT) (@) *to pause and any key to get it to continue.*

You can put DATA lines wherever you want in the program. Run each of these programs. They all work the same.

```
10  DATA APPLES
20  DATA ORANGES
30  FOR X = 1 TO 3
40  READ F$
50  PRINT "F$ = :" F$
60  NEXT X
70  DATA PEARS
```

```
30  FOR X = 1 TO 3
40  READ F$
50  PRINT "F$ = :" F$
60  NEXT X
70  DATA APPLES
80  DATA ORANGES
90  DATA PEARS
```

```
10  DATA APPLES, ORANGES
20  DATA PEARS
30  FOR X = 1 TO 3
40  READ F$
50  PRINT "F$ = :" F$
60  NEXT X
```

```
30  FOR X = 1 TO 3
40  READ F$
50  PRINT "F$ = :" F$
60  NEXT X
70  DATA APPLES, ORANGES,
    PEARS
```

Now Have It Build Your Vocabulary

Here are some words and definitions to learn:

```
                Words          Definitions

10   DATA TACITURN, HABITUALLY UNTALKATIVE
20   DATA LOQUACIOUS, VERY TALKATIVE
30   DATA VOCIFEROUS, LOUD AND VEHEMENT
40   DATA TERSE, CONCISE
50   DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
```

Now get the computer to select one of these words at random. Hmmm . . . there are ten items. Maybe this works:

```
60   N = RND(10)
70   FOR X = 1 TO N
80   READ A$
90   NEXT X
100   PRINT "THE RANDOM WORD IS:" A$
```

Run the program a few times. It doesn't work quite right. The computer's just as likely to stop at a definition as at a word.

What the computer really needs to do is pick a random word only from items 1, 3, 5, 7, or 9. Fortunately, BASIC has a word that helps with this. Type:

```
65   IF INT(N/2) = N/2 THEN N = N - 1
```

Now run the program a few times again. This time, it should work.

INT tells the computer to look at only the "whole part" of the number and ignore the decimal part. For instance, the computer sees INT(3.9) as 3.

Assume N, the random number, is 10. The IF clause in Line 65 does this:

```
INT(10/2) = 10/2
INT(5) = 5
5 = 5
```

The above is true: 5 does equal 5. Since it's true, the computer completes the THEN clause. N is adjusted to equal `9 (10 - 1)`.

Now assume N, the random number, is 9. The IF clause in Line 65 does this:

```
INT(9/2) = 9/2
INT(4.5) = 4.5
4 = 4.5
```

The above is not true: 4 does not equal 4.5. Since it's not true, the computer doesn't complete the THEN clause. N remains 9.

Besides reading a random word, the computer also must read the word's definition. Add these lines to the end of the program:

```
110  READ B$
120  PRINT "THE DEFINITION IS :" B$
```

Now run the program a few times.

Have the computer print one random word and definition after the next. Add this to the start of the program:

```
5  CLEAR 100
```

This reserves plenty of "string space." Add these lines to the end of the program:

```
130  RESTORE
140  GOTO 60
```

This lets the computer pick a new random word and its definition from a "restored" group of data items.

Here's how the program now looks:

```
5  CLEAR 100

10  DATA TACITURN, HABITUALLY UNTALKATIVE
20  DATA LOQUACIOUS, VERY TALKATIVE
30  DATA VOCIFEROUS, LOUD AND VEHEMENT
40  DATA TERSE, CONCISE
50  DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
60  N = RND(10)
65  IF INT(N/2) = N/2 THEN N = N - 1
70  FOR X = 1 TO N
80  READ A$
90  NEXT X
100  PRINT "A RANDOM WORD IS :" A$
110  READ B$
120  PRINT "ITS DEFINITION IS :" B$
130  RESTORE
140  GOTO 60
```

If you like, add some more words and definitions by adding DATA lines.

For variations on this program, you might try states and capitals, cities and countries, foreign words and meanings.

DO-IT-YOURSELF PROGRAM 8-3

Want to complete this program? Program it so that the computer:

1. Prints the definition *only*.
2. Asks you for the word.
3. Compares the word with the correct random word.
4. Tells you if your answer is correct. If your answer is incorrect, prints the correct word.

Here's our program:

```
5   CLEAR 500
10   DATA TACITURN, HABITUALLY UNTALKATIVE
20   DATA LOQUACIOUS, VERY TALKATIVE
30   DATA VOCIFEROUS, LOUD AND VEHEMENT
40   DATA TERSE, CONCISE
50   DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
60   N = RND(10)
65   IF INT(N/2) = N/2 THEN N = N - 1
70   FOR X = 1 TO N
80         READ A$
90   NEXT X
110   READ B$
120   PRINT "WHAT WORD MEANS :" B$
130   RESTORE
140   INPUT R$
150   IF R$ = A$ THEN 190
160   PRINT "WRONG"
170   PRINT "THE CORRECT WORD IS :" A$
180   GOTO 60
190   PRINT "CORRECT"
200   GOTO 60
```

Feel free to add frills such as a good-looking screen format or sound.

Learned in Chapter 8

BASIC WORDS

DATA
READ
RESTORE
INT
CLEAR

Notes

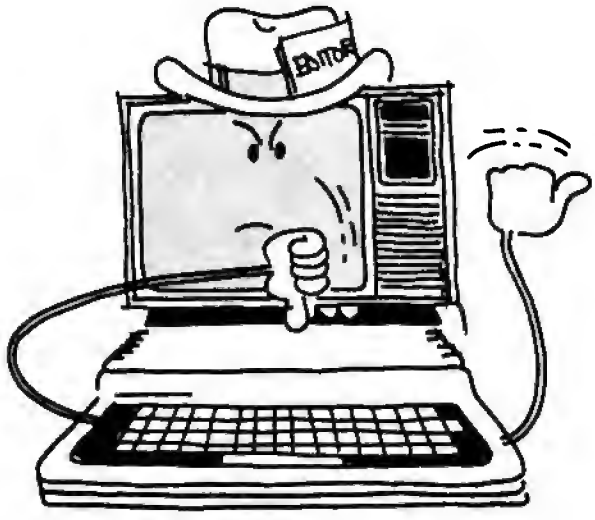
CHAPTER 9

WRITING

Up to now, you've probably been changing programs the long and boring way—by retyping them. If so, you'll be glad you've arrived at this chapter. You'll learn a new, easy way to change programs—by "editing" them.

Don't Throw Away That Line . . . Edit It!

(EDIT)



If you have a Deluxe Color Computer, EDIT will not work for you. You have a better way of editing program lines — the (ALT) *key. The* (ALT) *key is described in* Introducing Your Deluxe Color Computer.

Pretend you make a mistake typing a program. Line 50 somehow ends up:

```
50 DABA EFFFUSIVE, GIMPY MUSHY
```

You can change this line the hard way, by retyping it — *or* the easy way, by editing it. To get into Line 50's "edit mode," type:

```
EDIT 50 (ENTER)
```

You see:

```
50 DABA EFFFUSIVE, GIMPY MUSHY
50
```

You're now in the edit mode. While in this mode, you can use any of the special "edit keys" to display or change Line 50. They're all listed later in this chapter (Table 9.1).

Start by pressing (L), the edit key for "list." The (L) key displays the entire line again and then puts you back at the start.

MOVE ON DOWN THE LINE (CURSOR MOVEMENT)

Press (SPACEBAR) a few times. This key moves you forward. To move backward, press (←). Note that while in the edit mode (←) merely backspaces; it doesn't delete characters.

Move to the start of Line 50 and press (5) (SPACEBAR). This moves you *five* spaces forward — *all at once*. Do the same with (→). Press a number, such as (3), and (←) and move that many spaces backward.

Move to the start of Line 50 and press (S) (for "search") and then (E) (the character for which you want to search). This moves you to the first E. Move back to the start and press (2) (S) (E). This moves you to the *second* E in Line 50.

CHANGE THE LINE (CHANGE)

Make your first change to Line 50. Change DABA to DATA:

> *Once you enter the edit mode, you don't have to press (ENTER) after subcommands such as change, insert, list, and so on.*

Move to the "wrong" character — the B in DABA.

Press (C) for "change."

Type the new character, in this case, T.

To be sure the change is made, press (L) and you see:

```
50 DATA EFFFUSIVE, GIMPY MUSHY
```

Now make the next change: Change GIMPY to GUSHY. This time you'll change *three* characters at a time:

Move to the first wrong character — the I in GIMPY.

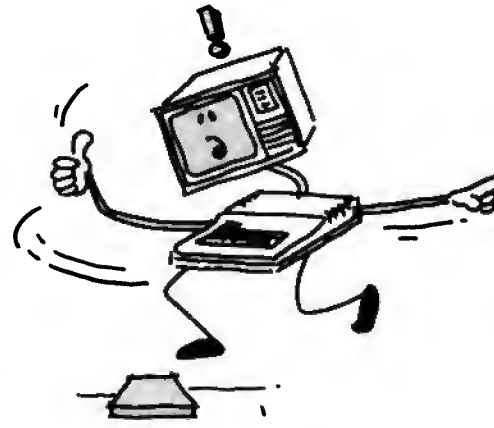
Press (3) (C) for "change three characters."

Type the three new characters — USH

Line 50 is now:

```
50 DATA EFFFUSIVE, GUSHY MUSHY
```

If this were all you needed to do to Line 50, you could press (ENTER) and get out of the edit mode. As you can see, though, you have much more work to do.



YOU'RE OUT! (DELETE)

You need to delete a character — one of the F's in EFFFUSIVE:

Move to the offensive character — the third F in EFFFUSIVE.

Press (D) for "delete."

And it's done. To confirm this, press (L) again:

```
50 DATA EFFUSIVE, GUSHY MUSHY
```

When we say "characters," we mean "spaces" too.

You can delete more than one character at a time. For example, if you press (4) (D), you'll delete four characters at a time.

SQUEEZE IT ALL IN (INSERT)

You now need to insert some characters: GUSHY should be *DEMONSTRATIVE OR* GUSHY.

Move to where you want to insert characters — the space before the G in Gushy.

Press (I) for "insert mode."

Type your insert — DEMONSTRATIVE OR

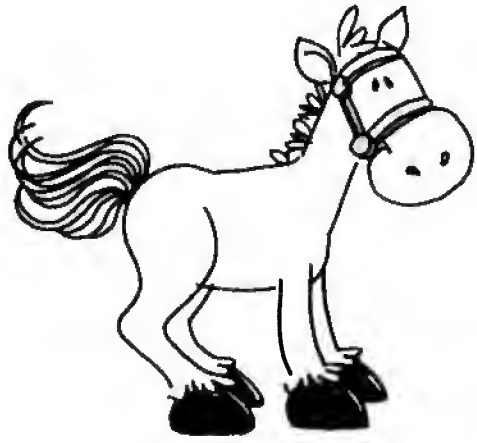
If you press (L) to list the line while using insert, you'll insert the letter "L" into the program line instead of listing the line.

At this point, you're still in the insert mode. For example, if you press (SPACEBAR), you'll insert a blank space; if you press (L), you'll insert an L. Therefore, you need to:

Press (SHIFT)(↑) to get out of the insert mode.

Now you can press (L) to list the line:

```
50 DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
   MUSHY
```



HACKAMORE OR HACKALESS? (HACK)

With "hack" you alter (halter?) a line by hacking the end of it and inserting new characters. Try hacking at Line 50:

Move to the first character you want hacked off — the M in MUSHY.

Press (H) for hack. This hacks off the rest of the line and puts you in the insert mode.

Type your insert — in this case, type CRUSTY.

Press (SHIFT)(↑) to get out of the insert mode.

If you list the line now (by pressing (L)), you see:

```
50 DATA EFFUSIVE, DEMONSTRATIVE OR GUSHY
   CRUSTY
```

KILL THE . . . AH . . . MISTAKE (KILL)

Kill is almost the opposite of hack. It "kills" everything up to the nth occurrence of a character. Suppose that, just for kicks, you want to kill the first half of Line 30 — everything up to the comma. Move to the start of Line 50 and press these keys:

(K),

If you list Line 50 now, you see:

```
50 ,DEMONSTRATIVE OR GUSHY CRUSTY
```

EXTENDED COLOR BASIC STRIKES AGAIN! (EXTEND)

Perhaps you want to "extend" Line 50:

Press (X) for extend. The cursor moves to the end of and you enter the insert mode.

Type your insert: AND MUSHY

Press (SHIFT)(←) to get out of the insert mode.

Line 50 is now:

```
50 ,DEMONSTRATIVE OR GUSHY CRUSTY AND MUSHY
```

Table 9.1/ Edit Keys

(*n* is a number. If you omit *n*, BASIC uses 1.)

Key	Action
(L)	Lists the line and moves to the start.
n(C)*characters*	Changes the next *n* characters to new *characters*.
(I)	Inserts characters.
n(D)	Deletes n characters.
(H)	"Hacks"' the rest of the line and puts you in the insert mode.
(X)	Lets you extend the line
n(S)*character*	Searches for the *n*th occurrence of *character*.
(K)	Kills rest of line.
n(K)*character*	Kills (deletes) up to the *n*th occurrence of *character*.
n(SPACEBAR)	Moves *n* spaces forward.
n(←)	Moves *n* spaces backward.

Mass Delete
(DELETE)

Up to now, you've deleted lines the simple way, like this:

```
50 (ENTER)
```

This works fine for one or two lines, but what if you want to delete 50 or 60 lines? You may find it easier to start over.

Extended Color BASIC comes to the rescue again with an easy way to delete program lines — the DEL command. For instance, if you want to delete Lines 30-50, type:

```
DEL 30-50 (ENTER)
```

Your Number's Up!
(RENUM)

So now you can change everything about a program line except the line number itself. Well, despair no more, because you can even do that with RENUM.

To see how RENUM works, type this small program:

```
10 PRINT "THIS IS THE FIRST LINE"
20 PRINT "THIS IS THE SECOND LINE"
30 PRINT "HERE'S ANOTHER LINE"
40 GOTO 10
```

Now renumber it. Type:

```
RENUM 100 (ENTER)
```

List the program and you see the new line numbers beginning with 100. Line 100 is what we call the *newline*:

```
100 PRINT "THIS IS THE FIRST LINE"
110 PRINT "THIS IS THE SECOND LINE"
120 PRINT "HERE'S ANOTHER LINE"
130 GOTO 100
```

Notice that even the GOTO line number reference is renumbered.

Renumber the program again with a *newline* of 200. Type:

```
RENUM 200,120 (ENTER)
```

Here, the *newline* is 200, but the renumbering *starts* with Line 120. Line 120 is what we call the *startline*:

```
100 PRINT "THIS IS THE FIRST LINE"
110 PRINT "THIS IS THE SECOND LINE"
200 PRINT "HERE'S ANOTHER LINE"
210 GOTO 100
```

Renumber the program one more time giving it an *increment* of 50 between each line:

```
RENUM 300,,50 (ENTER)
```

Here the *newline* is 300. Since you omitted the *startline*, BASIC renumbers the entire program. The *increment* between the lines is 50:

```
300 PRINT "THIS IS THE FIRST LINE"
350 PRINT "THIS IS THE SECOND LINE"
400 PRINT "HERE'S ANOTHER LINE"
450 GOTO 300
```

Here is the "syntax" of the RENUM command:

RENUM *newline, startline, increment*
Renumbers a program.
newline is the first new renumbered line. If you omit *newline*, BASIC uses 10.
startline is where the renumbering starts. If you omit *startline*, BASIC renumbers the entire program.
increment is the increment between each renumbered line. If you omit *increment*, BASIC uses 10.

Note: RENUM does not rearrange the order of lines.

Try some other variations of this command. Type:

```
RENUM ,,20
```

This renumbers your entire program. The *newline* is 10, and the *increment* is 20:

```
10 PRINT "THIS IS THE FIRST LINE"
30 PRINT "THIS IS THE SECOND LINE"
50 PRINT "HERE'S ANOTHER LINE"
70 GOTO 10
```

Type RENUM 40,30, (ENTER). Here, the *newline* is 40; the *startline* is 30; and the *increment* is 10:

```
10 PRINT "THIS IS THE FIRST LINE"
40 PRINT "THIS IS THE SECOND LINE"
50 PRINT "HERE'S ANOTHER LINE"
60 GOTO 10
```

Type RENUM 5,40 (ENTER) and you get a ?FC Error. This is because the result would move Line 40 ahead of Line 10.

Learned in Chapter 9

BASIC WORDS

EDIT
DEL
RENUM

Notes

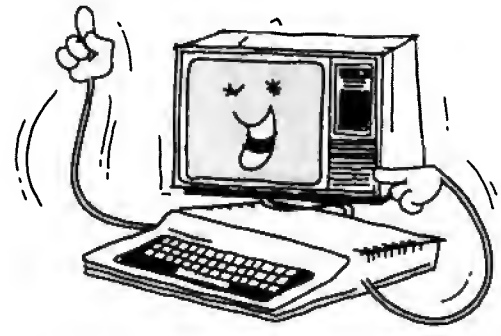
ARITHMETIC

Solving long math problems fast and accurately is a task your computer does with ease. Before typing long, difficult formulas, though, there're some shortcuts you'll want to use.

An easy way to handle complicated math formulas is with "subroutines." Type and run this program:

```
10  PRINT "EXECUTING THE MAIN PROGRAM"
20  GOSUB 500
30  PRINT "NOW BACK IN THE MAIN PROGRAM"
40  END

500  PRINT "EXECUTING THE SUBROUTINE"
510  RETURN
```

$$Ax\,(BY + C) - D + E\,(G/W) - F$$



GOSUB 500 tells the computer to go to the subroutine that starts at Line 500. RETURN tells the computer to return to the BASIC word that immediately follows GOSUB.

Delete Line 40 and see what happens when you run the program.

.....

If you did this, your screen shows:

```
EXECUTING THE MAIN PROGRAM
EXECUTING THE SUBROUTINE
NOW BACK IN THE MAIN PROGRAM
EXECUTING THE SUBROUTINE
?RG ERROR IN 510
```

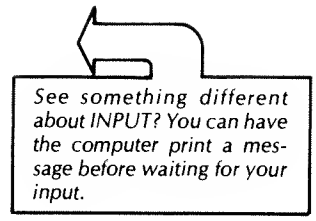
RG means "RETURN without GOSUB." Do you see why deleting END in Line 40 causes this error?

At first, the program runs just as it did before. It goes to the subroutine in Line 500 and then returns to the PRINT line that immediately follows GOSUB.

Then, since you deleted END, it goes to the next line—the subroutine in Line 500. This time, though, it doesn't know where to return. This is because it's merely "dropping" into the subroutine; it is not being sent to the subroutine by a GOSUB line.

This subroutine raises a number to any power:

```
10   INPUT "TYPE A NUMBER"; N
20   INPUT "TYPE THE POWER YOU WANT IT RAISED
     TO"; P
30   GOSUB 2000
40   PRINT : PRINT N "TO THE POWER OF" P "IS" E
50   GOTO 10
2000   REM FORMULA FOR RAISING A NUMBER TO A
       POWER
2010   E = 1
2020   FOR X = 1 TO P
2030   E = E * N
2040   NEXT X
2050   IF P = 0 THEN E = 1
2060   RETURN
```



Also introduced in this program are:

The colon (:), in Line 40. You can combine program lines using the colon to separate them. Line 40 contains the two lines: PRINT and PRINT N "TO THE" P "POWER IS" E.

REM, in Line 2000. REM means nothing to the computer. Put REM lines wherever you want in your program to help you remember what the program does; they make *no difference in the way the program works*. To see for yourself, add these lines and run the program:

PRINT by itself tells the computer to skip a line.

```
5   REM   THIS IS A PECULIAR PROGRAM,
17   REM   WILL THIS LINE CHANGE THE PROGRAM?
45   REM   THE NEXT LINE KEEPS THE SUBPROGRAM
     SEPARATED
```

DO-IT-YOURSELF PROGRAM 10-1

Change the above program so that the computer prints a table of squares (a number to the power of 2) for numbers, say, from 2 to 10.

The answer's in the back.

Give the Computer a Little Help

As math formulas get more complex, your computer needs help understanding them. For example, what if you want the computer to solve this problem:

Divide the sum of 13 + 3 by 8

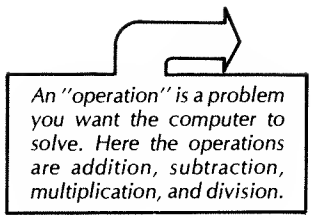
You may want the computer to arrive at the answer this way:

13 + 3 / 8 = 16/8 = 2

But, instead, the computer arrives at another answer. Type this command line and see:

```
PRINT 13 + 3 / 8 (ENTER)
```

The computer solves problems logically, using its own rules:



An "operation" is a problem you want the computer to solve. Here the operations are addition, subtraction, multiplication, and division.

RULES ON ARITHMETIC

The computer solves arithmetic problems in this order:

1. First, it solves any multiplication and division operations.
2. Last, it solves addition and subtraction operations.
3. If there's a tie (that is, more than one multiplication/division or addition/subtraction operation), it solves the operations from left to right.

In the problem above, the computer follows its rules:

- First, it does the division (3/8 = .375)
- Then, it does the addition (13 + .375 = 13.375)

For the computer to solve the problem differently, you need to use parentheses. Type this line:

```
PRINT (13 + 3) / 8 ENTER
```

Whenever the computer sees an operation in parentheses, it solves that operation before solving any others.

COMPUTER MATH EXERCISE

What do you think the computer will print as the answers to each of these problems?

```
PRINT 10 - (5 - 1) / 2 _____
PRINT 10 - 5 - 1 / 2 _____
PRINT (10 - 5 - 1) / 2 _____
PRINT (10 - 5) - 1 / 2 _____
PRINT 10 - (5 - 1 / 2) _____
```

Finished? Type each of the command lines to check your answers.

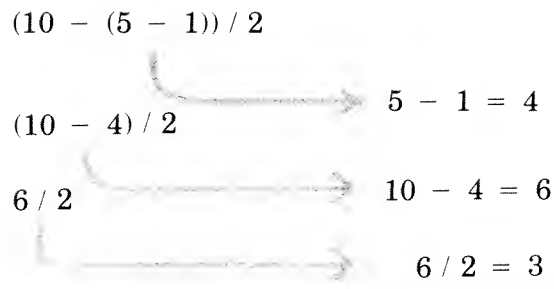
What if you want the computer to solve this problem?

Divide 10 minus the difference of 5 minus 1 by 2

You're actually asking the computer to do this:

(10 - (5 - 1)) / 2

When the computer sees a problem with more than one set of parentheses, though, it solves the inside parentheses and then moves to the outside parentheses. In other words, it does this:



RULES ON PARENTHESES

1. The computer solves operations enclosed in parentheses first, before solving any others.
2. The computer solves the innermost parentheses first. It then works its way out.

COMPUTER MATH EXERCISE

Insert parentheses in the problem below so that the computer prints 28 as the answer:

```
PRINT 30 - 9 - 8 - 7 - 6
```

Answer:

```
PRINT 30 - (9 - (8 - (7 - 6)))
```

Saving Routines

The program below uses two subroutines. It's for those of you who save by putting the same amount of money in the bank each month:

```
10   INPUT "YOUR MONTHLY DEPOSIT"; D
20   INPUT "BANK'S ANNUAL INTEREST RATE"; I
30   I = I/12 * .01
40   INPUT "NUMBER OF DEPOSITS"; P
50   GOSUB 1000
60   PRINT "YOU WILL HAVE $" FV "IN" P "MONTHS"
70   END

1000   REM COMPOUND MONTHLY INTEREST FORMULA
1010   N = 1 + I
1020   GOSUB 2000
1030   FV = D * ((E - 1) / I)
1040   RETURN
2000   REM FORMULA FOR RAISING A NUMBER TO A
       POWER
2010   E = 1
2020   FOR X = 1 TO P
2030   E = E * N
2040   NEXT X
2050   IF P = 0 THEN E = 1
2060   RETURN
```

Notice that one subroutine "calls" another. This is fine with the computer as long as:

there's a GOSUB to send the computer to each subroutine, and

there's a RETURN at the end of each subroutine.

Turn to the Appendix, "Subroutines." You'll find useful math subroutines you can add to your programs.

Learned in Chapter 10

BASIC WORDS	BASIC SYMBOLS	BASIC CONCEPTS
GOSUB	( )	Order of operations
RETURN	:	REM

Notes

CHAPTER 11

WORDS, WORDS, WORDS . . .

A great skill of the computer is its gift with words. It can tirelessly twist and combine words any way you want. With this gift, you can get it to read, write, and even talk.

Combining Words

Type and run this program:

```
10  PRINT "TYPE A SENTENCE"
20  INPUT S$
30  PRINT "YOUR SENTENCE HAS " LEN(S$) "
    CHARACTERS"
40  INPUT "WANT TO TRY ANOTHER"; A$
50  IF A$ = "YES" THEN 10
```

Impressed? LEN(S$) computes the length of string S$—your sentence. The computer counts each character in the sentence, including spaces and punctuation marks.

Erase the program and run this, which composes a poem (of sorts):

```
10  A$ = "A ROSE"
20  B$ = " "
30  C$ = "IS A ROSE"
40  D$ = B$ + C$
50  E$ = "AND SO FORTH AND SO ON"
60  F$ = A$ + D$ + D$ + B$ + E$
70  PRINT F$
```

Here the plus sign (+) combines strings. For example, D$ ("IS A ROSE") is a combination of B$ + C$.

There are two problems you may encounter when combining strings. Add the following line and run the program. It shows both problems:

```
80  G$ = F$ + F$ + F$ + F$ + F$ + F$ + F$
```

When the computer gets to Line 80, it prints the first problem with this line: ?OS ERROR IN 80 ("out of string space").

You will not get the OS error if you have not started up your computer since you ran the program from Chapter 8 with the CLEAR 500 line.

On startup, the computer reserves only 200 characters of space for working with strings. Line 80 asks it to work with 343 characters. To reserve room for this many characters and more (up to 500), add this line to the start of the program and run:

```
5  CLEAR 500
```

Now when the computer gets to Line 80, it has enough string space, but prints the second problem with this line: ?LS ERROR IN 80 ("string too long").

A string can contain no more than 255 characters. When storing more than 255 characters, you need to put these characters into several strings.

Twisting Words

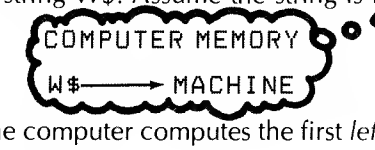
Now that you can combine strings, try to take a string apart. Type and run this program:

```
10  INPUT "TYPE A WORD"; W$
20  PRINT "THE FIRST LETTER IS : " LEFT$ (W$,1)
30  PRINT "THE LAST 2 LETTERS ARE : " RIGHT$
    (W$,2)
40  GOTO 10
```

Not impressed? Later, we'll show practical uses of this unusual skill.

Here's how the program works:

In Line 10 you input string W$. Assume the string is MACHINE:



In Lines 20 and 30, the computer computes the first *left* letter and the last two *right* letters of the string:

```
                M A C H I N E
LEFT$ (W$,1)                    RIGHT$ (W$,2)
```

Run the program a few more times to see how it works.

Now add this line to the program:

```
5  CLEAR 500
```

so that your computer will set aside plenty of space for working with strings. Run the program again. This time input a sentence rather than a word.

PROGRAMMING EXERCISE

How would you change Lines 20 and 30 so that the computer will give you the first 5 letters and the last 6 letters of your string?

20 _____

30 _____

Answers:

```
20  PRINT "THE FIRST FIVE LETTERS ARE :" LEFT$
    (W$,5)
```

```
30  PRINT "THE LAST SIX LETTERS ARE :" RIGHT$
    (W$,6)
```

Erase your program and type this one:

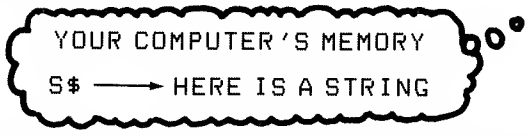
Remember how to erase a program? Type:
NEW (ENTER)

```
10  CLEAR 500
20  INPUT "TYPE A SENTENCE"; S$
30  PRINT "TYPE A NUMBER FROM 1 TO " LEN(S$)
40  INPUT X
50  PRINT "THE MIDSTRING WILL BEGIN WITH
    CHARACTER " X
60  PRINT "TYPE A NUMBER FROM 1 TO " LEN(S$) - X
    + 1
70  INPUT Y
80  PRINT "THE MIDSTRING WILL BE" Y
    "CHARACTERS LONG"
90  PRINT "THIS MIDSTRING IS :" MID$(S$,X,Y)
100  GOTO 20
```

Run this program a few times to see if you can deduce how MID$ works.

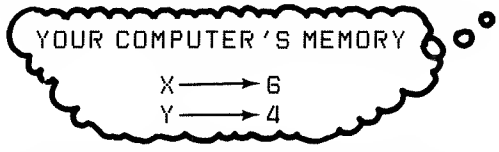
Here's how the program works:

In Line 20, assume you input HERE IS A STRING:



In Line 30, the computer first computes the length of S$, which is 16 characters. It then asks you to choose a number from 1 to 16. Assume you choose 6.

In Line 60, the computer asks you to choose another number from 1 to 12 (16-6+1). Assume you choose 4.



In Line 90, the computer gives you a "mid-string" of S$ that starts at the 6th character and is four characters long:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
H E R E   I S   A    S  T  R  I  N  G

          ←4→
          MID$(S$,6,4)
```

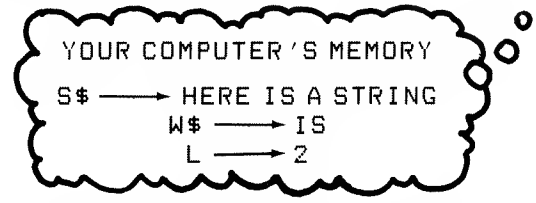
For another example of MID, erase the program and run this:

You can use this kind of program to sort through information. For instance, by separating strings, you could look through a mailing list for TEXAS addresses.

```
10  INPUT "TYPE A SENTENCE"; S$
20  INPUT "TYPE A WORD IN THE SENTENCE"; W$
30  L = LEN(W$)
40  FOR X = 1 TO LEN(S$)
50  IF MID$(S$,X,L) = W$ THEN 90
60  NEXT X
70  PRINT "YOUR WORD ISN'T IN THE SENTENCE"
80  END
90  PRINT W$ "--BEGINS AT CHARACTER NO." X
```

Here's how the program works:

In Line 20, assume you input the word IS for W$. In Line 30, the computer counts W$'s length: 2 characters.



In Lines 40-90 (the FOR/NEXT loop), the computer counts each character in S$, starting with character 1 and ending with character LEN(S$), which is 16.

Each time the computer counts a new character, it looks at a new mid-string. Each mid-string starts at character X and is L (2) characters long.

For example, when X equals 1, the computer looks at this mid-string:

```
 1
 H E R E   I S   A   S T R I N G
←2→
MID$(S$,1,2)
```

The fourth time through the loop, when X equals 4, the computer looks at this mid-string:

```
      4
H E R E   I S   A   S T R I N G
    ←2→
    MID$(S$,4,2)
```

When X equals 6, the computer finally finds IS, the mid-string for which it is searching.

DO-IT-YOURSELF PROGRAM 11-1

Start with a one-line program:

```
10  A$ = "CHANGE A SENTENCE."
```

Add a line that inserts this to the start of A$:

```
IT'S EASY TO
```

Add another line that prints the new sentence:

```
IT'S EASY TO CHANGE A SENTENCE
```

This is our program:

```
10  A$ = "CHANGE A SENTENCE."
20  B$ = "IT'S EASY TO"
30  C$ = B$ + " " + A$
40  PRINT C$
```

DO-IT-YOURSELF PROGRAM 11-2

Add to the above program to make it:

Find the start of this mid-string:

A SENTENCE

Delete the above mid-string to form this new string:

IT'S EASY TO CHANGE

Add these words to the end of the new string:

ANYTHING YOU WANT

Print the newly formed string:

IT'S EASY TO CHANGE ANYTHING YOU WANT

HINT: To form the string IT'S EASY TO CHANGE, you need to get the *left* portion of the string IT'S EASY TO CHANGE A SENTENCE.

Answer:

```
10  A$ = "CHANGE A SENTENCE."
20  B$ = "IT'S EASY TO"
30  C$ = B$ + " " + A$
40  PRINT C$
50  Y = LEN ("A SENTENCE")
60  FOR X = 1 TO LEN(C$)
70  IF MID$ (C$,X,Y) = "A SENTENCE" THEN 90
80  NEXT X
85  END
90  D$ = LEFT$ (C$,X - 1)
100  E$ = D$ + "ANYTHING YOU WANT"
110  PRINT E$
```

This program is the basis of a "word processing" program—a popular program that cuts down typing expenses.

DO-IT-YOURSELF CHALLENGER PROGRAM

Write a program that:

Asks you to input a sentence.

Asks you to input (1) a phrase within the sentence to delete and (2) a phrase to replace it.

Prints the changed sentence.

This may take a while, but you have everything you need to write it. Our answer's in the back.

Learned in Chapter 11

BASIC WORDS

LEN
LEFT$
RIGHT$
MID$

BASIC String OPERATOR

+

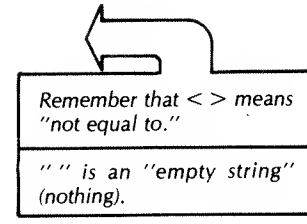
Notes

CHAPTER 12

A POP QUIZ

By using a word named INKEY$, you can get the computer to constantly "watch," "time," or "test" what you're typing. Type and run this program:

```
10  A$ = INKEY$
20  IF A$ <>"" GOTO 50
30  PRINT "YOU PRESSED NOTHING"
40  GOTO 10
50  PRINT "THE KEY YOU PRESSED IS---" A$
```



INKEY$ checks to see if you're pressing a key. It does this in a split second. At least the first 20 times it checks, you've pressed nothing (" ").

Line 10 labels the key you press as A$. Then the computer makes a decision:

If A$ equals *nothing* (" "), it prints YOU PRESSED NOTHING and goes back to Line 10 to check the keyboard again.

If A$ equals *something* (anything but " "), the computer goes to Line 50 and prints the key.

Add this line and run the program:

```
60  GOTO 10
```

No matter how fast you are, the computer is faster! Erase Line 30 to see what keys you're pressing.

Beat the Computer

Type this program:

```
10  X = RND(4)
20  Y = RND(4)
30  PRINT "WHAT IS" X "+" Y
40  T = 0
50  A$ = INKEY$
60  T = T + 1
70  SOUND 128,1
80  IF T = 15 THEN 200
90  IF A$ = "" THEN 50
100  GOTO 10

200  CLS(7)
210  SOUND 180, 30
220  PRINT "TOO LATE"
```

Here's how the program works:

Lines 10, 20, and 30 have the computer print two random numbers and ask you for their sum.

Line 40 sets T to 0. T is a timer.

Line 50 gives you your first chance to answer the question—in a split second.

Line 60 adds 1 to T, the timer. T now equals 1. The next time the computer gets to line 60 it again adds 1 to the timer to make T equal 2. Each time the computer runs Line 60 it adds 1 to T.

Line 70's there just to make you nervous.

Line 80 tells the computer you have 15 chances to answer. Once T equals 15, time's up. The computer insults you with Lines 200, 210, and 220.

Line 90 says if you haven't answered yet the computer should go back and give you another chance.

The computer gets to Line 100 only if you do answer. Line 100 sends it back for another problem.

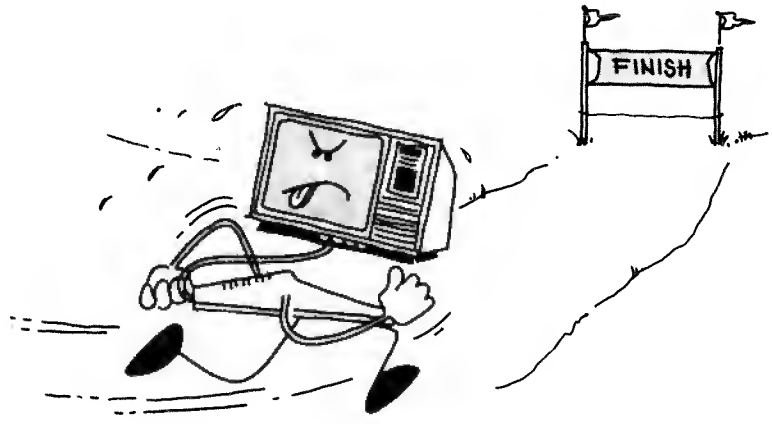
How can you get the computer to give you three times as much time to answer each question?

.....

Answer:

By changing this line:

```
80   IF T = 45 THEN 200
```



Checking Your Answers

How can you get the computer to check to see if your answer is correct? Would this work?

```
100   IF A$ = X + Y THEN 130
110   PRINT "WRONG", X "+" Y "=" X + Y
120   GOTO 10
130   PRINT "CORRECT"
140   GOTO 10
```

Remember the problem of mixing strings with numbers? Chapter 2 will refresh your memory.

If you run this program (and answer on time), you'll get this error message:

```
?TM ERROR IN 100
```

That's because you can't make a *string* (A$) equal to a *number* (X + Y). You somehow must change A$ to a number.

Change Line 100 by typing:

```
100  IF VAL(A$) = X + Y THEN 130
```

VAL(A$) converts A$ into its numeric value. If A$ equals the string "5," for example, VAL(A$) equals the number 5. If VAL(A$) equals the string "C," VAL(A$) equals the number 0. ("C" has no numeric value.)

To make the program more challenging, change these lines:

```
10  X = RND(49) + 4
20  Y = RND(49) + 4
90  B$ = B$ + A$
100  IF VAL(B$) = X + Y THEN 130
```

Then add these lines:

```
45  B$ = ""
95  IF LEN(B$) <> 2 THEN 50
```

A Computer Typing Test

Here's a program that times how fast you type:

```
10  CLS
20  INPUT "PRESS <ENTER> WHEN READY TO TYPE
    THIS PHRASE"; E$
30  PRINT "NOW IS THE TIME FOR ALL GOOD MEN"
40  T = 1
50  A$ = INKEY$
60  IF A$ = "" THEN 100
70  PRINT A$;
80  B$ = B$ + A$
90  IF LEN(B$) = 32 THEN 120
100  T = T + 1
110  GOTO 50
120  S = T/74
130  M = S/60
140  R = 8/M
150  PRINT
160  PRINT "YOU TYPED AT--"R"--WDS/MIN"
```

Line 40 sets T, the timer, to 1.

Line 50 gives you your first chance to type a key (A$). If you're not fast enough, Line 60 sends the program to Line 100 and adds 1 to the timer.

Line 70 prints the key you typed.

Line 80 forms a string named B$. Each time you type a key (A$), the program adds this to B$. For example, if the first key you type is "N," then:

```
A$ = "N"
and
B$ = B$ + A$
B$ = "" + "N"
B$ = "N"
```

If the next key you type is "O," then:

```
A$ = "O"
and
B$ = B$ + A$
B$ = "N" + "O"
B$ = "NO"
```

If the third key you type is "W," then:

```
A$ = "W"
and
B$ = "NO" + "W"
B$ = "NOW"
```

We could have made this calculation in one line by using parentheses:

```
120 R=8/((T/74)/60)
```

When the length of B$ is 32 (the length of NOW IS THE TIME FOR ALL GOOD MEN), the program assumes you've finished typing the phrase and goes to Line 120 to compute your words per minute.

Lines 120, 130, and 140 compute your typing speed. They divide T by 74 (to get the seconds), S by 60 (to get the minutes). They then divide the eight words by M to get the words per minute.

How about a variation of this program—a speed-reading test?

Learned in Chapter 12

BASIC WORDS

INKEY$
VAL

Notes

CHAPTER 13

MORE BASICS

Before you're finished with the "basics," you need to know a few more words.

The first is STOP. Type and run this program:

```
10  A = 1
20  A = A + 1
30  STOP
40  A = A * 2
50  STOP
60  GOTO 20
```

The computer starts running the program. When it gets to Line 30, it prints:

```
BREAK IN 30
OK
```

You now can type a command line to see what's happening. For example, type:

```
PRINT A (ENTER)
```

The computer prints 2—A's value when the program's at Line 30. Now type:

```
CONT (ENTER)
```

The computer continues the program. When it gets to Line 50, it prints:

```
BREAK IN 50
```

Type:

```
PRINT A (ENTER)
```

This time the computer prints 4—A's value at Line 50.

Type CONT again, and the computer breaks again at Line 30. If you have it again print A, it prints 5—the value of A at Line 30 the second time through the program.

Inserting STOP lines in your program helps you figure out why it's not working the way you expect. When you fix the program, take the STOP lines out.

For Long Programs . . .

To save memory, you can omit spaces in your program before and after punctuation marks, operators, and BASIC words.

Clear memory and type:

```
PRINT MEM (ENTER)
```

The computer prints how much storage space remains in the computer's memory.

When you're typing a long program, you will want to have the computer PRINT MEM from time to time to make sure you're not running out of memory.

Help with Typing

Type this program:

```
10  INPUT "TYPE 1, 2, OR 3"; N
20  ON N GOSUB 100, 200, 300
30  GOTO 10

100  PRINT "YOU TYPED 1"
110  RETURN

200  PRINT "YOU TYPED 2"
210  RETURN

300  PRINT "YOU TYPED 3"
310  RETURN
```

Run it.

ON . . . GOSUB in Line 20 works the same as three lines:

```
18  IF N = 1 THEN GOSUB 100
20  IF N = 2 THEN GOSUB 200
22  IF N = 3 THEN GOSUB 300
```

ON . . . GOSUB looks at the line number following ON—in this case N.

If N is *1*, the computer goes to the subroutine starting at the *first line number* following GOSUB.

If N is *2*, the computer goes to the subroutine starting at the *second line number*.

If N is *3*, the computer goes to the subroutine starting at the *third line number*.

What if N is 4? Since there's no fourth line number, the computer simply goes to the next line in the program.

Here is a program that uses ON . . . GOSUB:

```
5  FOR P = 1 TO 600: NEXT P
10  CLS: X = RND(100): Y = RND(100)
20  PRINT "(1) ADDITION"
30  PRINT "(2) SUBTRACTION"
40  PRINT "(3) MULTIPLICATION"
50  PRINT "(4) DIVISION"
60  INPUT "WHICH EXERCISE(1-4)"; R
70  CLS
```

```
80  ON R GOSUB 1000, 2000, 3000, 4000
90  GOTO 5

1000  PRINT "WHAT IS" X "+" Y
1010  INPUT A
1020  IF A = X + Y THEN PRINT "CORRECT" ELSE
      PRINT "WRONG"
1030  RETURN

2000  PRINT "WHAT IS" X "-" Y
2010  INPUT A
2020  IF A = X-Y THEN PRINT "CORRECT" ELSE
      PRINT "WRONG"
2030  RETURN

3000  PRINT "WHAT IS" X "*" Y
3010  INPUT A
3020  IF A = X*Y THEN PRINT "CORRECT" ELSE
      PRINT "WRONG"
3030  RETURN

4000  PRINT "WHAT IS" X "/" Y
4010  INPUT A
4020  IF A = X/Y THEN PRINT "CORRECT" ELSE
      PRINT "WRONG"
4030  RETURN
```

When A does not equal X + Y, the condition set up in Line 1020 is not true.

Notice the word ELSE in Lines 1020, 2020, 3020, and 4020. You can use ELSE if you want the computer to do something special when the condition is not true. In Line 1020, *if* your answer—A—equals X + Y, *then* the computer prints CORRECT or *else* it prints WRONG.

You may use ON . . . GOTO in a similar way as ON . . . GOSUB. The only difference is that ON GOTO sends the computer to another line number rather than to a subroutine.

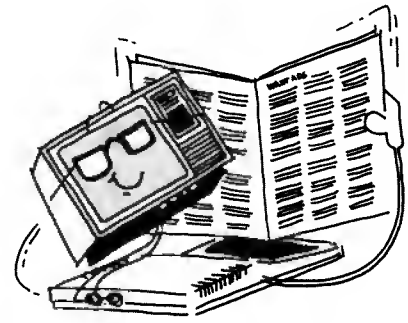
Here's part of a program using ON . . . GOTO:

```
10  CLS
20  PRINT @ 134, "(1) CRAZY EIGHTS"
30  PRINT @ 166, "(2) 500"
40  PRINT @ 198, "(3) HEARTS"
50  PRINT @ 354, "WHICH DO YOU WANT TO PLAY"
60  INPUT A
65  CLS
70  ON A GOTO 1000, 2000, 3000

1000  PRINT @ 230, "CRAZY EIGHTS GAME"
1010  END

2000  PRINT @ 236, "500 GAME"
2010  END

3000  PRINT @ 235, "HEARTS GAME"
3010  END
```

Does the Job Say "AND" or "OR"?

Anyone who speaks English knows the difference between "and" and "or"—even your computer. For example, assume there's a programming job opening. The job requires:

A degree in programming
AND
Experience in programming



Erase memory and type:

```
10  PRINT "DO YOU HAVE--"
20  INPUT "A DEGREE IN PROGRAMMING"; D$
30  INPUT "EXPERIENCE IN PROGRAMMING"; E$
40  IF D$ = "YES" AND E$ = "YES" THEN PRINT "YOU
    HAVE THE JOB" ELSE PRINT "SORRY, WE CAN'T
    HIRE YOU"
50  GOTO 10
```

Run the program. You may answer the questions this way:

```
DO YOU HAVE--
A DEGREE IN PROGRAMMING? NO
EXPERIENCE IN PROGRAMMING? YES
SORRY, WE CAN'T HIRE YOU
```

Now, assume the requirements change so that "or" becomes "and." The job now requires:

A degree in programming
OR
Experience in programming

To make this change in the program, type:

```
40  IF D$ = "YES" OR E$ = "YES" THEN PRINT
    "YOU'VE GOT THE JOB" ELSE PRINT "SORRY, WE
    CAN'T HIRE YOU"
```

Run the program and see what a difference AND and OR makes:

```
DO YOU HAVE--
A DEGREE IN PROGRAMMING? NO
EXPERIENCE IN PROGRAMMING? YES
YOU HAVE THE JOB
```

More Arithmetic

These words can save many program lines:

SGN

SGN tells you whether a number is positive, negative, or zero:

```
10  INPUT "TYPE A NUMBER"; X
20  IF SGN(X) = 1 THEN PRINT "POSITIVE"
30  IF SGN(X) = 0 THEN PRINT "ZERO"
40  IF SGN(X) = -1 THEN PRINT "NEGATIVE"
50  GOTO 10
```

Run the program, inputting these numbers:

```
15   -30   -.012   0   .22
```

ABS

ABS tells you the absolute value of a number (the magnitude of the number without respect to its sign). Type:

```
10  INPUT "TYPE A NUMBER"; N
20  PRINT "ABSOLUTE VALUE IS" ABS(N)
30  GOTO 10
```

Run the program inputting the same numbers as the ones above.

STR$

STR$ converts a number to a string. Example:

```
10  INPUT "TYPE A NUMBER"; N
20  A$ = STR$(N)
30  PRINT A$ + " IS NOW A STRING"
```

Exponents

Type and run this program to see how the computer deals with very large numbers:

```
10  X = 1
20  PRINT X;
30  X = X * 10
40  GOTO 20
```

Notice the OV (overflow) error at the end. The computer can't handle numbers larger than 1E+38 or smaller than -1E+38. (It rounds off numbers around 1E-38 and -1E-38 to 0.)

The computer prints very large or very small numbers in "exponential notation." "One billion" (1,000,000,000), for example, becomes 1E+09, which means "the number 1 followed by nine zeros."

*Or technically 1*109, which is 1 times 10 to the ninth power: 1*10*10*10*10*10*10*10*10*10*10*10*

If an answer comes out "5E-06," you must shift the decimal point, which comes after the 5, six places to the left, inserting zeroes as necessary. Technically, this means 5*10-6, or 5 millionths (.000005).

In our BASIC, that's 5/10/10/10/10/10/10

Exponential notation is simple once you get used to it. You'll find it an easy way to keep track of very large or very small numbers without losing the decimal point.

Congratulations, Programmer!

You've now learned the "basics" and can no doubt write some decent programs. The next section will help you add excitement to your programs with graphics and music.

Learned in Chapter 13

BASIC WORDS		BASIC SYMBOLS	BASIC CONCEPT
STOP	SGN	AND	Exponential notation
CONT	ABS	OR	
MEM	STR$		

Notes

SECTION II

SIGHTS AND SOUNDS

Have you reached your fill of BASIC basics? In this section, you'll take a dramatic leap and learn to:

- Draw a circle
- Paint a house
- Compose a song
- Cool off with a cube
- And much more!

And you'll also be amazed at how quickly and easily you can do this! So turn the page and we'll get right to the point.

CHAPTER 14

LET'S GET TO THE POINT

One of the most exciting features of Extended Color BASIC is its ability to display precise, varied, and easy-to-use graphics called "high-resolution graphics."

Just how easy-to-use are these graphics? Well, let's start with the most basic (pun intended) graphic element—a dot (or point)—and build from there.

A 10,600-mile journey starts with a single step, and even the Mona Lisa began with a single stroke on the canvas. (A Jackson Pollock might begin with a single splatter!)

Extended Color BASIC makes it simple to put a dot on the screen. Type the following program and see:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 PSET (10,20,8)
40 GOTO 40
```

Don't worry about any of the new words. PMODE and SCREEN, for instance, determine the degree of detail and the range of color. They are covered in later chapters.

Now run the program. The screen should be buff, and if you look carefully, you can see a small orange dot in the upper left corner. That dot was put there by the PSET (point set) in Line 30.

PSET lets you set a dot anywhere on the screen. It has this format:

PSET (*h,v,c*) sets a point on the current graphics screen

h is the horizontal position (0 to 255).
v is the vertical position (0 to 191).
c is the color (0 to 8). If you omit *c*, BASIC uses the current foreground color.

You'll see these "syntax blocks" throughout this section. They'll help you understand the "parameters" you can use with graphic statements.

Even though you can't see it, the computer has divided your screen into a grid of nearly 50,000 dots—256 across and 192 down—so that you can put a dot precisely where you want it. Simply look up the dot's position in the Graphics Screen Worksheet in the back of this manual.

Look at Line 30 again and see how PSET specifies the dot's position (10 over and 20 down):

```
30 PSET (10,20,8)
```

Here's the statement you would use to set an orange dot in the center of the screen:

```
PSET (128,96,8)
```

Now add a program line that sets an orange dot in the lower right corner (255 over and 191 down).

Is this the line you used?

```
35 PSET (255,191,8)
```

If so, congratulations! You've made your point. Run your program and you'll see.

Now list the program. It should look like this:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 PSET (10,20,8)
35 PSET (255,191,8)
40 GOTO 40
```

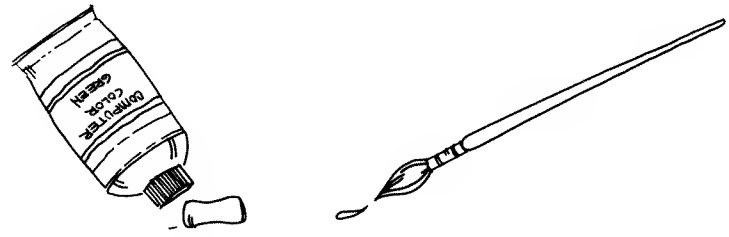
You're off to a great start . . .

Very Important Note! The Color Computer can produce 9 colors: black, green, yellow, blue, red, buff, cyan, magenta, and orange. The actual shade you get, though, depends on your T.V.—not the computer. We suggest you perform the color adjustment test program in your introduction manual before running these programs.

. . . But What About the Color?

By now, you've probably figured out that you can change colors by changing *c* to a different number in the range 0 to 8.

Within limits, this is true. However—and it's a big however—you may not get the color you specified. There's a good reason for this, which we'll cover later in the discussion of the different graphic "modes." For now, don't worry if you don't always get the color you want.



Here is the list of color codes:

Code	Color
0	Black
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

The 8 gives the color (orange). Later, we'll discuss how to change the color. For now, simply use orange.

If you want to try changing the dots' colors, use buff (5), cyan (6), or magenta (7). Then change the color back to orange (8) before proceed-

ing. (These 4 colors are the only ones available with your current program.)

Now You See It . . . Now You Don't

Any guesses how to turn off a dot? Here's a hint: It's easy and it has to do with color.

You see, you don't really turn off the dot, you simply change its color so that it blends into the background. You do this with a new statement: PRESET (point reset). PRESET "knows" you want to use the background color, so you don't need to give the color.

PRESET (*h,v*) resets a point on the current graphics screen

h is the horizontal position (0 to 255).
v is the vertical position (0 to 191).

DO-IT-YOURSELF PROGRAM 14-1

Get to know the dot positions on your TV screen by using your Graphics Screen Worksheet.

Select several points on the worksheet, identify them in terms of their (X,Y) coordinates, and display them on the screen using the program we used to get you started. Don't change any program lines except those that contain PSET(*h,v,c*).

DO-IT-YOURSELF PROGRAM 14-2

Do you remember the RND (random) function from Section I? If not, review it; then write a short program that fills the screen with random dots of random colors.

The Last Point

Before you finish this chapter, we want to make one more point. You can use PPOINT to find out what color any dot on the screen is.

PPOINT (*h,v*) tells what color a point is on the current graphics screen

h is the point's horizontal position (0 to 255).
v is the point's vertical position (0 to 191).

This example shows how PPOINT can be handy to include in a program:

```
5  PMODE 3,1
10  PCLS
15  SCREEN 1,1
30  X = RND(10)
35  Y = RND(10)
40  C = RND(8)
50  PSET (X,Y,C)
60  IF PPOINT (5,5)=8 THEN GOTO 105
70  GOTO 30
105  CLS
110  PRINT @ 100, "POSITION (5,5)
     IS NOW ORANGE"
```

The computer fills a 10X10 "square" (in the upper left corner of the screen) with random colored dots. When the dot in Position (5,5) is filled with an orange dot (Code 8), the computer displays the message POSITION (5,5) IS NOW ORANGE.

Learned in Chapter 14

BASIC WORDS	CONCEPTS
PSET	Setting points
PRESET	Resetting points
	Changing colors
PPOINT	Finding a point's color

Notes

CHAPTER 15

HOLD THAT LINE!

So you can put a dot on the screen—even several dots. But what kind of starting point is that, you may wonder, when you're eager to create some "real" graphics.

To answer that question, think of some of your very first "drawings" on paper. Perhaps they were detailed pictures of clowns and trained seals and other wonderful things. How did you draw such marvels? Probably by connecting a bunch of dots.

And that is exactly how your computer "draws." You tell it which dots to connect, and it draws a line.

That's Some Line You Have

One way to tell the computer to draw a line between dots is to use the Extended Color BASIC statement LINE. To see LINE at work, modify the program that set the dots. (For the sake of convenience, call the program "Lines.")

First change Line 30 as follows:

```
30 LINE (0,0) - (255,191),PSET
```

Then delete Line 35 by typing:

```
35 (ENTER)
```

Your program should now read:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 LINE (0,0)-(255,191),PSET
40 GOTO 40
```

Now run the program. The screen should display an orange line that runs from the upper left to the lower right on a buff background.

How about changing the direction of the line so that it runs from the lower left to the upper right?

You've probably already figured this one out, but—just in case—here's the new Line 30:

```
30 LINE (0,191)-(255,0),PSET
```

X Marks the Spot

What about intersecting lines?

Using the Graphics Screen Worksheet, plot the points used in creating the intersecting lines in the "Lines" program.

Reinsert the original Line 30 that drew the first line. (First, renumber it as Line 25.) Then run the program. Does your screen display 2 orange lines intersecting in the center?

In fact, you can put as many lines on the screen as you want—once you learn the format. Here it is:

LINE ***(h1,v1)-(h2,v2),a,b*** draws a line or a box on the current graphics screen

(h1,v1) is the line's start point.
(h2,v2) is the line's end point.
a is either PSET (set) or PRESET (reset).
b is either B (box) or BF (box filled). This is optional.

Note: You may omit the start point as discussed below.

Just as in the old dot-to-dot days, you may often want to draw a line that begins at the last line's end point. Whenever this is the case, you may omit the start point. The computer automatically starts at either the end point set by the latest LINE statement or—if you haven't yet used LINE in the program—at (128,96). Here is an example:

```
30 LINE (0,0)-(255,191),PSET
35 LINE -(191,0),PSET
```

Line 20 draws a line from (0,0) to (255,191). Line 30 then draws another line, this one from (255,191) to point (191,0).

Regardless of whether or not you include the start point, you must precede the end point with a hyphen (-).

How About Dropping a Line?

We've discussed the line's start and end points. Now let's turn to the next parameter in the LINE statement—PSET or PRESET.

Take another look at the program lines that created the intersecting lines:

```
30 LINE (0,0)-(255,191),PSET
35 LINE (0,191)-(255,0),PSET
```

From your experience with turning on and off dots in Chapter 14, do you have any idea what the PSET parameter is doing and what would happen if you change it to PRESET? Try it and see. Change the PSET in Line 25 to PRESET and run the program again:

```
30 LINE (0,0)-(255,191),PRESET
```

The color specification is elsewhere in the program, not in the LINE command, so we'll come back to it later. For now, just concentrate on LINE.

If you guessed that the orange line that ran from the upper left to the lower right would "disappear," you were right.

Now replace PSET in Line 30 with PRESET. The screen went blank, right? The reason is the way PSET and PRESET work in a LINE statement:

PSET sets the line in the pre-specified foreground color.

PRESET sets the line back to the pre-specified background color so that you can't see it.

Note: The PSET and PRESET parameters in a LINE statement are not the same as the PSET and PRESET statements discussed in Chapter 14. They do not specify a dot or a color code. They merely specify that the line be set to the foreground or the background color.

Before proceeding, change the PRESET parameters in Lines 25 and 30 back to PSET.

To B (a Box) or Not to B . . .

We've almost made it through LINE, but a few items still need to be (to B?) covered.

B stands for "box."

With Extended Color BASIC, you can make a box without having to write a separate program line for each side. All you have to do is specify two opposing corners of the box and add ,B to the statement. Then, when you run the program, your computer creates a box instead of a line.

To illustrate this, call your "Lines" program back into service.

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

As is, the program creates 2 orange lines that intersect in the center of the screen. Delete Line 30 and add the suffix ,B to Line 25. Now see what happens when you run the program.

```
25 LINE (0,0)-(255,191),PSET,B
```

Did you box yourself in?

DO-IT-YOURSELF PROGRAM 15-1

Write a program that creates a box with a pair of lines intersecting in the center. We'll tell you why these are the only available colors when we discuss PMODE and SCREEN in the next 2 chapters.

Fill It Up

We're almost at the end of the LINE, so let's try to finish.

If you refer to the format of LINE, you can see you have the option of adding F to the optional suffix ,B.

F lets you "fill" the box with the foreground color. Try it. Change Line 25 as follows:

```
25 LINE (0,0)-(255,191),PSET,BF
```

How about that! You should have a big orange box (256 x 192) on a buff background.

That's Color with a Capital C, Capital O, Capital . . .

In Chapter 14, we explained how to use the *c* parameter of the PSET command to change the color of a dot. But we've also been talking for some time about foreground and background colors. Now it's time to explain them further.

Naturally, if you're using one color heavily, you don't want to have to specify it each time you put something on the screen. With the COLOR feature, you don't have to.

Within certain limits, the graphics feature COLOR lets you set the foreground/background colors. (See "PMODE" and "SCREEN" later in this book.) Here is its format:

COLOR *foreground,background* sets the foreground and background color on the current graphics screen

foreground is the code (0 to 8) for the foreground color.
background is the code (0 to 8) for the background color.

Note: As stated in Chapter 14, the only colors available with your current program are buff (5), cyan (6), magenta (7), and orange (8).

When you don't specify the foreground and background colors, your computer automatically chooses the highest-numbered available color code for the foreground color and the lowest-numbered available color code for the background color. That's why the crossing lines in the "Lines" program are orange (8) on a buff background (5).

To see COLOR in action, call on "Lines" again:

```
5 MODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
```

```
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

Insert Line 6 into your program:

```
6 COLOR 5,7
```

Now run the program. What do you think of buff lines crossing on a magenta background?

If you used your Color Computer to draw an airplane and used COLOR to give it the right color, would you have flying colors?

Do you want to see what the colors look like when reversed? If so, retype or edit the line like this:

```
6 COLOR 7,5
```

In the next chapter, you'll learn how to make even more colors available.

DO-IT-YOURSELF PROGRAM 15-2

Ready to try your own "Lines" program? Can you build a house? Start with Lines 5, 10, and 20 of the "Lines" program and take it from there. Be sure to add:

A front door, of course.

At least one window. (Don't forget to turn the lights on or off.)

A chimney. (You won't need a chimneysweep, not yet anyway!)

The overall design is up to you (Cape Cod, Ranch, or whatever), but we've included a sample house (good view, no pets) program in the back of the book. Don't worry about doorknobs; we'll add those later.

Be sure to save this program on cassette, since you'll be needing it later. (You'll find it much easier to draw the house if you plot its points on a Graphics Screen Worksheet.)

DO-IT-YOURSELF PROGRAM 15-3

This should be a real challenge for you.

As you know, a straight line is the shortest distance between two points. Well, put a few extra miles between our two points. Use LINE to draw a crooked line.

To get started, use Lines 5, 10, and 20 from the "Lines" program.

COLOR is not an action statement; it must precede an action statement (such as PCLS or LINE) before the foreground and background colors are actually changed.

Learned in Chapter 15

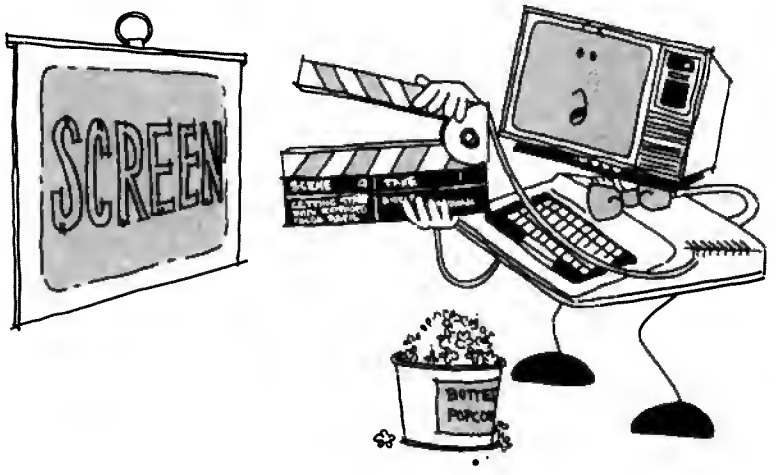
BASIC WORDS	CONCEPTS
LINE	Drawing a line Erasing a line Drawing a box Filling in a box
COLOR	Changing foreground and background colors

Notes

CHAPTER 16

THE SILVER SCREEN

Are you ready to find out about another statement? If so, turn down the lights and butter the popcorn, because we're about to raise the curtain on the silver screen.



A Word About Video Memory

Whenever you want to display an image on your TV, the computer stores the screen image in "video memory." The computer's TV-circuitry then "reads" the screen image and displays it on your TV.

The "normal" video memory is large enough for text (letters and numbers) but not for graphics (circles, lines, boxes, and so on). Consequently, the computer has two video memories: one for text and one for graphics.

Lighting the Silver Screen

Take a look at our "Lines" program for a second. Concentrate on the SCREEN statement in Line 20:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

SCREEN tells the computer to display a screen image on your TV. What kind of screen it displays depends on the instructions you give it:

First, you tell the computer whether to use the TV screen for text (such as letters or numbers) or graphics (such as lines and circles).

Second, you tell the computer what "color set" to use.

SCREEN *type, color set* displays the current graphics or text screen

type is 0 (text screen) or 1 (graphics screen)
color set is 0 or 1

Note: If *type* or *color set* is any positive number greater than 1, your computer uses 1.

In the "Lines" program, change Line 20 to:

```
20 SCREEN 0,0
```

Then run the program. Does your computer "hang up"? (Press **BREAK** to regain control.)

Any time your program outputs text (PRINT, INPUT), the computer automatically performs a SCREEN 0,0 command. In a "2-color mode," described in the next chapter, this gives you a black and green screen.

Actually, the computer ran "Lines," just as before. This time, though, it did not show you the graphics screen. You asked to see the text screen instead.

Now change Line 20 to:

```
20 SCREEN 1,0
```

Notice that you have the graphics screen again, but this time the color set has been changed.

At first glance, it appears that you have only 2 color choices—0 and 1. Actually, though, you're choosing from a much greater variety: You're switching color sets, not individual colors.

Color Set 0	Green/Yellow/Blue/Red
Color Set 1	Buff/Cyan/Magenta/Orange

DO-IT-YOURSELF PROGRAM 16-1

Do you understand SCREEN? If you do, write a program that switches from text screen to graphics screen. You might want to put a loop in the program so that it changes the color set after it loops through the program. This way you can see all the SCREEN features at work.

Clearing the Silver Screen
(PCLS)

Your "Lines" program should look like this:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

Look at Line 10. It contains the PCLS statement. This statement simply clears the graphics screen. (It serves the same function for the graphics screen as CLS does for the text screen.)

Here is the syntax for PCLS:

PCLS *color* clears the current graphics screen

color is 0-8. If you omit the *color*, the computer clears the screen to the current background color.

The "Lines" program doesn't make use of PCLS's *color* option. Therefore, the computer uses the current background color, buff. Retype Line 10:

```
10 PCLS 6
```

Run the program. Your screen now displays orange lines on a cyan background.

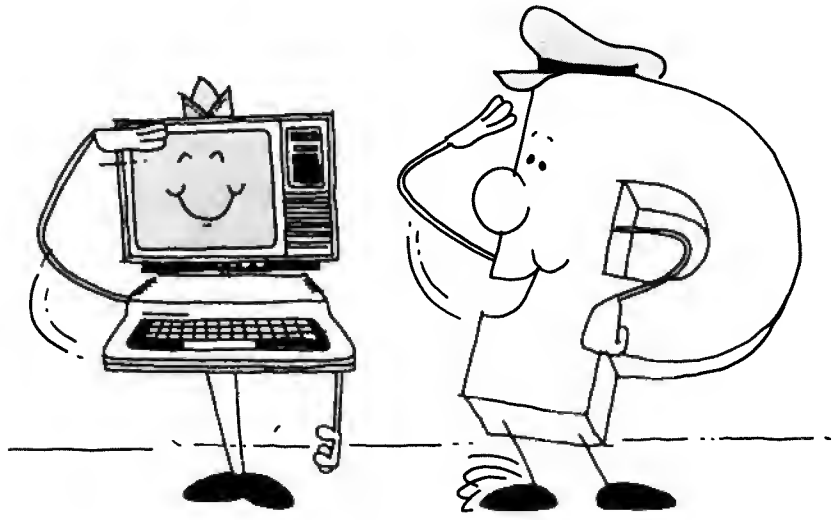
Learned in Chapter 16

BASIC WORDS	CONCEPTS
SCREEN	Displaying the current screen
PCLS	Clearing the graphics screen

Notes

MINDING YOUR PMODES

What lets you produce exciting graphics is the massive size of graphics memory. To get a perspective on this, contrast graphics and text memory: Text memory has 512 memory locations; graphics memory has up to 12,288!



You can use the power of graphics memory in three ways:

To produce graphics with very *high resolution* (fine detail).

To produce graphics with *many colors.*

To produce fast-changing, "animated" graphics by retaining *many graphics screens in memory* at once.

How much you can use of each of these features depends on how you "set" graphics memory. The more you use of one feature—such as retaining many screens in memory—the less you can use of the other features (high resolution and colors).

PMODE—the unknown statement in the "Lines" program—is what sets the features you want to use. PMODE lets you set 5 "modes," shown in Table 17-1. Each mode, of course, has its own trade-off of features.

Table 17-1/ PMODE Settings

	Resolution	Colors	Screens
PMODE 4	high	2	2
PMODE 3	medium	4	2
PMODE 2	medium	2	4
PMODE 1	low	4	4
PMODE 0	low	2	8

"Lines" in Mode 4

Bring back "Lines" and see what it looks like in a different mode. In case you've forgotten "Lines," here it is:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

Now change from Mode 1 to Mode 4.

```
5 PMODE 4,1
```

Run the program. You should spot two feature changes right away:

The color changes because you shifted from a 4-color mode to a 2-color mode.

The lines are much finer because they're in high resolution.

(The next chapter talks about the third feature; the one having to do with storing more than one graphics screen in memory.)

Colors à la Mode

A 2-color mode, just like a 4-color mode, has 2 color sets. You saw one of the 2-color sets—black and buff—when you ran "Lines" in Mode 4. To see "Lines" in the other 2-color set—black and green—make this change:

```
20 SCREEN 1,0
```

Table 17-2 shows what color sets you can use in 2-color and 4-color modes.

Think of when you first started drawing. You probably used wide crayons. When you got better, you began using thin crayons so that you could draw thin lines—lines with better "resolution."

Table 17-2/ Color Sets

	2-Color	4-Color
SCREEN 1,0	Black/Green	Green/Yellow/Blue/Red
SCREEN 1,1	Black/Buff	Buff/Cyan/Magenta/Orange

"Lines"—Through Thick and Thin

Notice that when you ran "Lines" in high resolution (Mode 4), you didn't have to change any dot positions. Color BASIC uses the same 256 x 192 screen grid, no matter what the resolution is.

For example, (128,96) is *always* the center of the screen, no matter what resolution you're using, and (0,0) is always the upper-left corner of the screen.

The size of each dot on the screen, though, is different in each resolution:

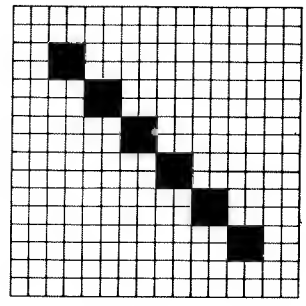
Low resolution uses *four* grid dots to set a screen dot. When the computer sets Dot (0,0), for example, it also sets (1,0), (1,1), and (0,1).

Medium resolution uses *two* grid dots to set a screen dot. When the

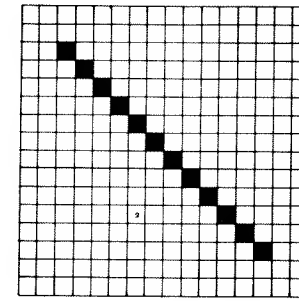
computer sets Dot (0,0), it also sets (1,0).

High resolution uses only *one* grid dot to set a screen dot. When the computer sets Dot (0,0), that's all it sets.

Thus, a diagonal line in low resolution looks more like a stairstep than one drawn in high resolution:



Low resolution



High resolution

And the number of *different* screen positions you can use in low resolution is only one-fourth what you can use in high resolution (see Table 17-3).

Table 17-3/ Graphics Screen Resolution

	Screen Positions Available	Size of Each Dot
High resolution	256 x 192	
Medium resolution	128 x 192	
Low resolution	128 x 96	

Here is a program that shows a box cycle through each mode. Notice that with each mode the box's lines go from thick to thin and its colors go from 2 colors to 4 colors.

```
5 FOR MODE = 0 TO 4
10 PMODE MODE,1
20 PCLS
30 SCREEN 1,1
40 LINE (75,50)-(125,100),PSET,B
50 FOR Y = 0 TO 500: NEXT Y
60 NEXT MODE
70 GOTO 5
```

This is PMODE's format. The next chapter shows how to use use the second parameter, *start page*.

PMODE *mode,start page* sets the current graphics screen in graphics memory

mode specifies the features you want to use in graphics memory. If you omit mode, the computer uses the last *mode* or (if none) Mode 2.

start page specifies on which page in graphics memory to start a graphics screen. If you omit *start page*, the computer uses the last *start page* or (if none) Page 1.

Therefore, if you omit PMODE, the computer uses PMODE 2,1.

Keep in mind that the graphics screen is always full of "dots." The issues are simply how many, what size, and what color.

Learned in Chapter 17

BASIC WORDS	CONCEPT
PMODE	Selecting a resolution mode Selecting color availability

Notes

FINDING THE RIGHT PAGES

In writing this book, we've "stored" chapters on pages. Some chapters require more pages; some less.

In the same sense, Color BASIC stores graphics screens on 1,536-byte blocks of graphics memory called "pages." Some screens require more pages; some less.

Table 18-1 shows how many pages it takes to draw a screen in each mode. As you can see, a screen drawn in a higher mode (which offers higher resolution or more colors) consumes more memory pages than a screen drawn in a lower mode.

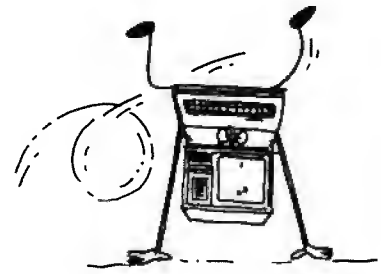


Table 18-1/ Pages Required for Graphics Screens

Screen	Pages Required
Mode 4 Screen	4 pages
Mode 3 Screen	4 pages
Mode 2 Screen	2 pages
Mode 1 Screen	2 pages
Mode 0 Screen	1 page

See what happens if you store the now famous (infamous) "Lines" screen on different pages.

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

Focus on PMODE. As you know, the first PMODE parameter tells the computer to start a Mode 1 screen. And, as Table 18-1 tells you, a Mode 1 screen requires two pages.

The second parameter tells the computer to start the screen on Page 1. Thus, the 2-page "Lines" screen is on Pages 1 and 2.

To put the 2-page "Lines" screen on Pages 3 and 4, type:

```
5 PMODE 1,3
```

Run the program. This shows the same screen, but the screen is in on entirely different pages.

How about storing two screens—one on Pages 1 and 2, and another on Pages 3 and 4? Change Line 5, delete Line 20, and add Lines 27 and 28. What you end up with is this:

```
5 PMODE 1,1
10 PCLS                                  stores screen on
25 LINE (0,0)-(255,191),PSET             Pages 1-2

27 PMODE 1,3
28 PCLS                                  stores screen on
30 LINE (0,191)-(255,0),PSET             Pages 3-4

40 GOTO 40
```

The first part of the program starts a Mode 1 screen on Pages 1-2. It "clears" this screen and puts a line on it.

The next part of the program starts another Mode 1 screen on Pages 3-4. It clears this screen and puts a line on it.

Run the program and you won't see either screen, because there's no SCREEN statement. So add SCREEN:

```
35 SCREEN 1,1
```

Now run the program and you see one screen—the one stored on Pages 3-4.

Whenever Color BASIC displays a screen, it uses your most recent PMODE instruction to tell it what the "current graphics screen" is. In this case, the most recent PMODE—PMODE 1,3—tells Color BASIC that the current graphics screen is a Mode 1 screen on Pages 3-4.

You may have noticed that all the graphics statements (LINE, PPOINT, PSET, PRESET, PCLS, SCREEN, and COLOR) produce graphics on the "current graphics screen." The most recent PMODE statement is what sets the current graphics screen.

Insert another PMODE line just before SCREEN, and Color BASIC displays a Mode 1 screen on Pages 1-2:

```
32 PMODE 1,1
```

Just for kicks, have Color BASIC display a Mode 2 screen that starts on Page 2. Any guesses on what you'll see? Change Line 32 to PMODE 2,2 and run the program. Since Mode 2 requires two pages, you see what's on Pages 2-3. And, since this is Mode 2, you see this screen in 2 colors with medium resolution.

Flipping Screens

As you know, animators make cartoons by drawing many still pictures and then "flipping" through them.

So here's the moment you've been waiting for! This program flips screens to show two lines in motion:

Did you know that it takes more than 12,000 individual drawings to make just one 7-minute cartoon? Wouldn't a computer be a help there!

```
5 PMODE 1,1
10 PCLS                                  stores Page 1-2 screen
25 LINE (0,0)-(255,191),PSET

27 PMODE 1,3
28 PCLS                                  stores Page 3-4 screen
30 LINE (0,191)-(255,0),PSET

32 PMODE 1,1
34 SCREEN 1,1                            displays Page 1-2
36 FOR I=1 TO 200:NEXT I                 screen
```

```
38 PMODE 1,3
40 SCREEN 1,3
42 FOR I=1 TO 200:NEXT I
44 GOTO 32
```

displays Page 3-4 screen

Adding Pages

You can use a maximum of 8 pages of graphics memory—Pages 1-8. However, when you first start up, Color BASIC gives you only half that amount—Pages 1-4. For example, make this change to "Lines":

```
5 PMODE 1,4
```

Run "Lines" and you get a ?FC Error. You're asking Color BASIC to use Pages 4-5, but Page 5 is not available!

To remedy the problem, insert Line 4 and you now have all 8 pages.

```
4 PCLEAR 8
```

PCLEAR lets you reserve from 1 to 8 pages of memory. If you use PCLEAR, it needs to be your program's first or second statement (after CLEAR, if you use CLEAR):

PCLEAR *pages* reserves pages of graphics memory

pages is the amount of graphics memory to reserve (0-8)

On startup, the computer automatically reserves 4 pages. Use PCLEAR to reserve more or fewer pages.

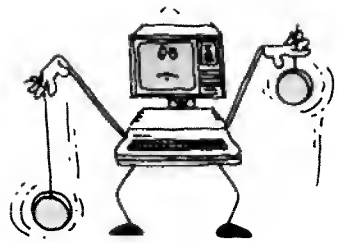
You may wonder why we don't use PCLEAR 8 all the time. The reason: PCLEAR 8 decreases program memory. Sometimes you need more *program* memory; other times you need more *graphics* memory. PCLEAR sets the balance.

If you ever have a conflict between program memory requirements and video memory requirements, you'll get a ?OM ERROR (Out of Memory).

Up and Down, Up and Down

You probably think your computer is a little crazy, but now we'll prove that it's a real yo-yo. In fact, you can call this program "Yo-Yo." Enter and run it.

```
10 PCLEAR 8
20 FOR P=1 TO 8
30 PMODE 0,P
40 PCLS
50 LINE (128,0)-(138,10+(P-1)*15),PSET
60 CIRCLE (128,P*15),15
70 NEXT P
80 FOR P=1 TO 8:GOSUB 110:NEXT P
90 FOR P=7 TO 1 STEP -2:GOSUB 110:NEXT P
100 GOTO 80
110 PMODE 0,P
120 SCREEN 1,0
130 FOR T=1 TO 10:NEXT T
140 RETURN
```



With the exception of CIRCLE (see the next chapter), you've already learned all the features used by this program.

PCOPY

Using PCOPY ("page copy") you can copy one page of graphics memory to another. Here is the format for PCOPY:

PCOPY *page1* TO *page 2* copies page1 to page2

For example, if you want to copy Page 3 to Page 8, type:

```
PCOPY 3 TO 8
```

One advantage of PCOPY is it can shorten your programs by eliminating repetition.

Keep in mind PCOPY copies one graphics' memory page. Unless you're in Mode 0, this is not one screen. For example, in Mode 4, the above statement copies only one-fourth of a screen.

DO-IT-YOURSELF PROGRAM 18-1

The following program displays 4 squares that are on 4 different memory pages on the screen at the same time. Run it, and then shorten the program using PCOPY.

```
4 PCLEAR 8
5 PMODE 3,4
10 PCLS
11 SCREEN 1,1
12 LINE (110,20)-(120,30),PSET,B
20 PMODE 3,3
21 SCREEN 1,1
22 LINE (110,20)-(120,30),PSET,B
30 PMODE 3,2
31 SCREEN 1,1
32 LINE (110,20)-(120,30),PSET,B
40 PMODE 3,1
41 SCREEN 1,1
42 LINE (110,20)-(120,30),PSET,B
50 GOTO 50
```

DO-IT-YOURSELF PROGRAM 18-2

Using LINE and *start page*, simulate a lightning storm. (Put "crazy lines" at random positions on different pages. Then switch back and forth between pages.)

Learned in Chapter 18

BASIC WORDS	CONCEPTS
PCLEAR	Reserving pages for graphics
PMODE	Selecting a *start page* Flipping pages to simulate motion
PCOPY	Copying graphics from one page to another

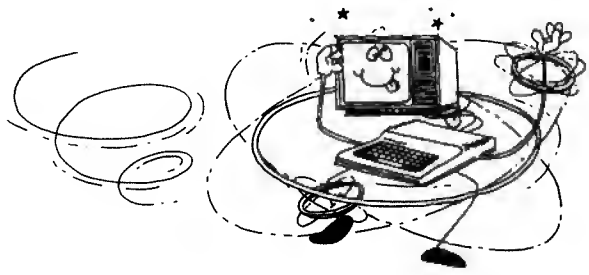
Notes

CHAPTER 19

GOING IN CIRCLES

Does all this talk about SCREEN, PMODE, and PCLEAR have you going in circles? If so, you haven't seen anything yet!

For example, you can create a full circle or ellipse, or a partial circle or ellipse using a single statement, CIRCLE. Here is the syntax of CIRCLE:



CIRCLE (*h,v*),*r*,*c*,*hw*,*start*,*end* draws a circle on the current graphics screen

h is the horizontal position of the centerpoint (0 to 255).
v is the vertical position of the centerpoint (0 to 191).
r is the radius in screen points.
c is any available color (0-8). If you omit *c*, the computer uses the foreground color.
hw is the height to width ratio (0 to 255). If you omit *hw*, the computer uses 1.
start is the starting point (0 to 1). If you omit *start*, the computer starts at 0.
end is the ending point (0 to 1). If you omit *end*, the computer uses 1.

If the *start* point is equal to the *end* point or if you omit both the *start* and the *end*, the computer draws the complete ellipse.

To draw a circle, you need only the centerpoint (*h,v*) and the radius (*r*), which is the distance from the center in points.

First, count over on the h-axis, then down on the v-axis to locate the desired center. Then, once you specify that point, indicate the circle's radius. The largest radius that fits on the screen is 95. If the radius is larger than 95, the circle "flattens" against the edges of the screen.

Bring your "Lines" program back into service.

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

Your program should read:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 CIRCLE (128,96)
   ,95
40 GOTO 40
```

Delete Line 25 and change Line 30 as follows:

```
30 CIRCLE (128,96),95
```

Run the program. Your TV should display a somewhat scruffy, orange circle on a buff background. Are you wondering why the circle isn't truly round? Look at Line 5 and you'll see; the computer is in Mode 1 (medium resolution).

Change Mode 1 to Mode 4 (high resolution) as follows:

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,1
30 CIRCLE (128,96),95
40 GOTO 40
```

Run the program. Now that's a circle! (It should be a buff circle on a black background.)

DO-IT-YOURSELF PROGRAM 19-1

Using the program above, generate a bull's eye. You can do this one of two ways:

Add a separate program line for each concentric circle but use a common center (*h,v* coordinate).

Use a FOR . . . NEXT loop with a STEP 10 to have the computer do the work for you.

DO-IT-YOURSELF PROGRAM 19-2

Do you still have the program for the house you built? How do you expect to get into the house without a doorknob? Use CIRCLE to put a doorknob on the front door. Your Graphics Screen Worksheet is helpful in locating the exact point you need.

Note: If you use medium or low resolution, a circle small enough to serve as a doorknob does not have much detail. Run the program in Mode 4 for more detail.

Coloring the Circle

After you decide on the circle's radius, choose its color. Using 2-color mode, you haven't much choice, but using 4-color mode (Mode 1 or 3), you'll find the color option an exciting feature.

Your program should read:

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 CIRCLE (128,96),95
40 GOTO 40
```

First, make the circle a more manageable size:

```
30 CIRCLE (128,96),30
```

Now, for a little variety, change the color to cyan:

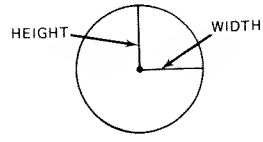
```
30 CIRCLE (128,96),30,6
```

It's as easy as that! In fact, you can change the circle's color to any of the available colors.

Putting on the Squeeze

Did you ever take a Hula-Hoop, bicycle tire, or buggy wheel and squeeze it with both hands to form an ellipse?

Similarly, you can change circle on your screen into an ellipse by using the height/width ratio (*hw*) option.



Notice that your CIRCLE statement does not include the color code. Omitting the code tells the computer to use the foreground color. You must include the comma, though, to indicate to the computer that you are omitting the c and that the number specifies the hw ratio.

The width of the ellipse is equal to the radius. The height is determined by *hw*. If *hw* is 1, the computer draws a circle. If *hw* is greater than 1, it draws an ellipse that is higher than it is wide. If *hw* is less than 1, it draws an ellipse that is wider than it is high. For example, this program draws a circle:

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,1
30 CIRCLE (128,96),30,,,1
40 GOTO 40
```

If however, you change *hw* as shown here, the program draws a vertical ellipse:

```
30 CIRCLE (128,96),30,,,3
```

If you change *hw* as shown here, it draws a horizontal ellipse:

```
30 CIRCLE (128,96),30,,,25
```

If *hw* equals 0, then the "ellipse" becomes "infinitely" wider than it is high. In other words, it becomes a horizontal line.

You could say the circle is finally on the straight and narrow path.

As *hw* increases past 1, the "ellipse" approaches a vertical line.

Change Line 30 in the following ways and run the program:

When you use 0, imagine you're looking at a coin from the edge, and you'll have a good idea of what we mean.

```
30 CIRCLE (128,96),30,,,0
```

```
30 CIRCLE (128,96),30,,,100
```

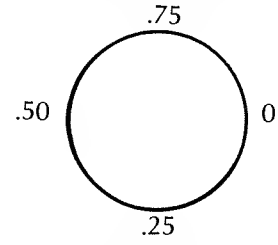
From Start to Finish . . .

Suppose you want to draw only part of a ellipse (an arc). To do this, you must list the ellipse's center point (*h*,*v*), its radius (*r*), and its height/width ratio (*hw*). If you wish, you may precede *hw* with the color (c).

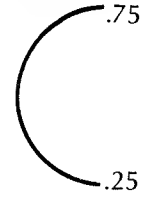
> **Note:** To draw an arc, you must specify *hw*. For a normal arc, use *hw* 1.

From the above information, the computer knows the location, width, and height of the ellipse. Now you can tell it how much of the ellipse to draw.

To do this, specify the *start* of the arc (0 to 1) and *end* (0 to 1) of the arc, following the chart below. Keep in mind that the computer always draws clockwise.



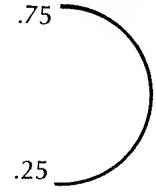
Suppose, for example, you want to draw this arc:



To do so, use this statement:

```
30 CIRCLE (128,96),30,1,,.25,,.75
```

Now change the statement to draw this arc:



Is this your new Line 30?

```
30 CIRCLE (128,96),30,1,,.75,,.25
```

DO-IT-YOURSELF PROGRAM 19-3

Has night fallen on the house you built? If so, you might want to shed some light on the subject by putting a crescent moon in the corner. This requires two intersecting arcs and some trial and error on your part.

DO-IT-YOURSELF PROGRAM 19-4

Maybe it's cold, as well as dark, around your house. If so, build a fire in the fireplace and show smoke coming out the chimney. (Use CIRCLE to generate a spiral that simulates the smoke.)

Learned in Chapter 19

BASIC WORDS	CONCEPTS
CIRCLE	Drawing a circle or an ellipse Coloring a circle or an ellipse Drawing an arc

Notes

THE BIG BRUSH-OFF

You might think we've forgotten this is a Color Computer. So far, it's been a little dab here and a splotch or two there. You'll never create a masterpiece that way! Well, it's time to loosen up a little and paint the town, if not red, then at least a bright orange.

The Extended Color BASIC graphics function PAINT lets you "paint" any shape with any available color.

Here is the syntax for PAINT:

PAINT (*h,v*),*c*,*b* paints the current graphics screen

h is the horizontal position (0 to 255) of the point at which painting is to begin.
v is the vertical position (0 to 191).
c is the color (0 to 8).
b is the border color at which painting is to stop (0 to 8).

If the computer reaches a border other than that of the specified color, it paints over that border.

Change the "Lines" program as follows:

```
5 PMODE 3,1
10 PCLS
20 SCREEN 1,1
30 LINE (0,0)-(255,191),PSET
40 LINE (0,191)-(255,0),PSET
50 CIRCLE (128,96),90
60 PAINT (135,125),8,8
70 GOTO 70
```

Before you run the program, can you predict the results? Lines 30 and 40 make the intersecting lines. Line 50 generates a circle the center of which is at the point where the two lines intersect. That part should be easy, but what about PAINT in Line 60?

If you guessed the computer goes to screen position (135,125) and paints with orange until the paint reaches an orange border, you're right!

Delete Line 30 and then run the program. Now that you redefine the borders, the computer paints half the circle.

DO-IT-YOURSELF PROGRAM 20-1

Can you paint the entire circle? You can do this two ways. One involves adding a line; the other involves deleting a line.

By the way, did you notice the computer's mode and color set? Mode 3 is a 4-color mode, and Color Set 1 gives you buff, cyan, magenta, and orange.

Stay in Mode 3, but change the color set (SCREEN 1,0) and run the program. Without changing any other lines, you should get a red circle (border) on a green background.

To avoid confusion about color, change the PAINT color to fit the color set:

```
60 PAINT (135,125),2,4
```

Now when you run the program, the semicircle should be painted yellow (Code 2) until the computer encounters the red (Code 4) border.

Remember, you can paint using only those colors that are available in your mode and color set.

But you didn't specify red lines and red paint! Do you have any idea what happened?

When the computer is in a 4-color mode and you specify a color it can't supply, the computer subtracts 4 from Codes 5 through 8. (It interprets 0 as 3.)

DO-IT-YOURSELF PROGRAM 20-2

Do you still have your house? It probably looks a little plain, maybe even shabby. Why don't you spruce it up with some paint?

DO-IT-YOURSELF PROGRAM 20-3

Add a garage to your house, then use PAINT to raise and lower the garage door. Since the painting action always goes up first, this takes a little refining on your part. Add a delay before and after the opening. (With CIRCLE, add the sun.) By the way, did you notice the computer's mode and color set? Mode 3 is a 4-color mode, and Color Set 1 gives you buff, cyan, magenta, and orange.

Learned in Chapter 20

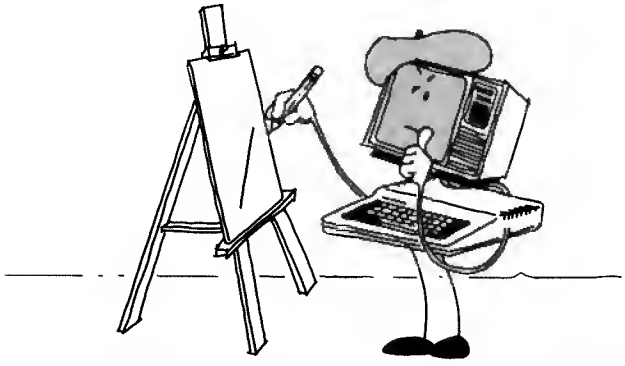
BASIC WORDS	CONCEPTS
PAINT	Painting any figure

Notes

CHAPTER 21

DRAW THE LINE SOMEWHERE

You already know how to create lines, ellipses, and boxes. Now how would you like to learn a shortcut for doing some of those things? The shortcut is DRAW, which lets you draw a line (or series of lines) by specifying direction, angle, and color—all in the same program line! Here is the syntax of DRAW:



DRAW *line* draws a shape on the current graphics screen

line is a string expression that may include the following motion commands, modes, and options:

Motion Commands

M = Move the draw position
U = Up
D = Down
L = Left
R = Right
E = 45-degree angle
F = 135-degree angle
G = 225-degree angle
H = 315-degree angle
X = Execute a substring and return

Modes

C = Color
A = Angle
S = Scale

Options

N = No update of draw position
B = Blank (no draw, just move)

Note: If *line* is a string constant, you must enclose it in quotes.

Always insert the B option directly before the M motion command; otherwise, unwanted lines may appear.

This program probably has replaced your dog as your best friend.

```
5 PMODE 3,1
10 PCLS
20 SCREEN 1,1
25 DRAW "BM128,96;
    U25; R25; D25;
    L25"
40 GOTO 40
```

To make the program easier to read, we've separated each motion statement with a semicolon (;). You needn't do this. You must, however, always separate the (h,v) coordinates with a comma (,).

Earlier you learned how to create a box using LINE. To do this, you may have had to do some difficult figuring with the Graphics Screen Worksheet to locate the necessary *start* and *end* points.

With DRAW, you have to locate only the *start* point and then tell the computer in which direction to draw and how far to do so. If you omit the *start* point, the computer starts at the last DRAW position or—if you haven't previously used DRAW—at the center of the screen.

Use your "Lines" program to try out DRAW. Delete Line 30 and change Line 25 to the following:

```
25 DRAW "BM128,96;U25;R25;D25;L25"
```

Presto! Can you guess why the square's lower left corner is at (128,96)? Look at the first two numbers inside the quotes.

The motion command, M, tells the computer at which point to later begin drawing.

M *h,v* tells the computer at which point to begin drawing

h is the horizontal position (0 to 255).
v is the vertical position (0 to 191).

Note: Always preface M by the letter B; if you do not, unwanted lines appear.

The above program tells the computer to start drawing at (128,96), draw up (U) 25 points, right (R) 25 more, down (D) 25 more points, and finally left (L) 25.

Note: If you omit the line's length, the computer uses 1 as the length.

Setting the Square on Edge (Diagonal Lines)

Instead of drawing horizontal and vertical lines, stand the square on one of its corners. To do this, substitute E, F, G, and H for U, R, L, and D in Line 25:

```
25 DRAW "BM128,96;E25;F25;G25;H25"
```

This DRAW starts at (128,96) too. Instead of going up, however, the first line angles off at 45 degrees; the computer draws the next 3 lines at their designated angles.

If you are in Mode 0 or 1 and use E, F, G, or H to generate a line that has an odd-number length and at least 1 odd-number coordinate (*h,v*), Lines F and H have a slight "hitch" at the midpoint. If both coordinates are even-numbered, Lines E and G have the "hitch." This is normal.

DO-IT-YOURSELF PROGRAM 21-1

You already know your computer is the star of the show, but can you prove it by drawing a star? Use the DRAW motion commands for both perpendicular and diagonal lines.

Absolute M *v* Relative M

Suppose you draw a square and then want to draw another one nearby. You know exactly how far away you want the second square to be, but don't want to have to locate the coordinates (*h,v*).

Absolute motion: "Go to the corner of 53rd Street and Bomber Lane."

Relative motion: "Go 2 blocks down, take a right, and go 1 more block."

Another form of the M command lets you specify "relative" motion instead of "absolute" motion. So far, you have used absolute motion; you have specified points in terms of their coordinates (*h,v*). Using relative motion, you can specify points in relation to the current point (the point last drawn).

Here's the syntax for relative motion:

Y *sign h-offset, v-offset* lets you specify points relative to the current point

h-offset is the distance to move horizontally from the current position. If you precede it with a plus sign (+), the h-position increments by the specified amount. If you precede it with a minus sign (-), the h-position decrements.

v-offset is the distance to move vertically from the current. If you precede *v-offset* with a plus sign (+) or if you omit the sign, the v-position increments by the specified amount. If you precede it with a minus sign (-), the v-position decrements.

For example, if you wish to create a second box at a position relative to that of the first box in the (redefined) "Lines" program, you might add this line:

```
30 DRAW "BM+15,15;U25;R25;D25;L25"
```

When the computer executes Line 30, the current draw position is (128,96), which is the the last draw position in Line 25. So the lower left corner of the new square is at (238+15,96+15) or (255,111).

Change Line 30 as follows:

```
30 DRAW "BM+15,-15;U25;R25;D25;L25"
```

Run the program. The *start* point of the new square is (128+15,96-15) or (143,81).

DO-IT-YOURSELF PROGRAM 21-2

After all this heated activity, you're probably ready to cool off, so why don't you use DRAW to create an ice cube?

You can generate the entire cube using DRAW, or you can incorporate a couple of LINE commands within the program. Try to use both absolute and relative motion.

Tipping the Scales

What if the figures you draw turn out to be too big or too small?

The solution's easy. Your computer has a built-in function that lets you "scale" (up or down) any display generated by DRAW. All you have to do

When you use the scale-down option, the computer rounds the resulting line length to the nearest whole number, if it is not already a whole number.

For example, "S2U25R25D25L25" results in a 12-1/2 x 12-1/2 square. The computer draws a 13 x 13 square.

is use the S*x* command in the string.

S*x* lets you scale a display

x is a number in the range 1 to 62 that indicates the scale factor in units of 1/4 as shown here:

1 = 1/4 scale
2 = 2/4 scale
3 = 3/4 scale
4 = 4/4 (full) scale
5 = 5/4 (125%) scale
8 = 8/4 (double) scale
12 = 12/4 (triple) scale
etc.

If you omit *x*, the computer uses 4 (4/4 = 1).

After you enter an S*x* command, the computer scales all absolute and relative motion commands accordingly until you enter another.

Make your refined "Lines" draw a single square again. Do this by deleting Line 30 and changing Line 25 as follows:

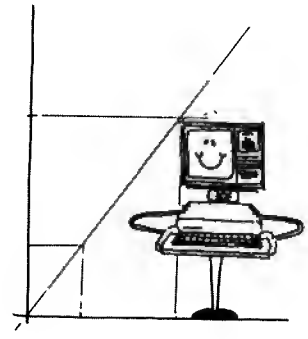
```
25 DRAW "S2;FM128,96;U25;R25;D25;L25"
```

Run the program. The square in the lower left corner should be half the size you specified.

To see how small or large a square can be, run the following program:

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,1
25 FOR SCALE = 1 TO 62
30 S$ = "S" + STR$(SCALE) + ";"
35 DRAW S$ + "BM10,100U20R20D20L20"
40 NEXT SCALE
50 GOTO 50
```

Don't make the mistake of thinking that the smallest square is the one specified in Line 35. The one we specified is the fourth one from the edge.



Color Me . . .

DRAW's C option lets you specify the color of a particular line.

First, list the "Lines" program:

```
5 PMODE 3,1
10 PCLS
20 SCREEN 1,1
30 DRAW "S2;BM128,96;U25;R25;D25;L25"
40 GOTO 40
```

Go back to full scale either by changing S2 to S4 or by deleting S2. Then, just inside the first set of quotation marks in Line 30, insert:

```
C6
```

Run the program. Does it display a cyan square on a buff background?

Replace the C6 (in program Line 30) with C8 and run the program. Did the square turn orange?

C must take the following form:

C*x* lets you specify a line's color

x is the color code (0 to 8). If you omit *x*, the computer uses the foreground color.

You can insert C*x* anywhere inside the DRAW statement. All actions that follow are the color you specify. For instance, change Line 30 to read:

```
30 DRAW "C8; BM128,96;U25;R25;
   C6; D25;L25"
```

Run the program. The program displays a 2-color square. The first 2 lines drawn are orange. The second 2 are cyan.

If you want to "erase" a line, draw another line on top of it using the background color.

What's Your Angle?

Another option that is available with DRAW is A. This option lets you specify the angle at which a line is to be drawn. After you include A in the DRAW command, the computer draws all subsequent lines with the angle displacement specified by A*x* until you specify otherwise.

Your program should now read:

```
5 PMODE 3,1
10 PCLS
20 SCREEN 1,1
30 DRAW "C6;BM128,96;U25;
   R25;D25;L25"
40 GOTO 40
```

Here is the syntax for the A command:

A*x* lets you specify the angle of a line

x is the angle code (0 to 3). All angles are measured clockwise.

0 = 0 degrees
1 = 90 degrees
2 = 180 degrees
3 = 270 degrees

If you omit A*x*, the the computer uses A0.

To illustrate this, change program Line 30 to read:

```
30 DRAW "A0;BM128,96;U25"
```

Run the program. Your screen displays a vertical line that is 25 points long. Now change Line 30:

```
30 DRAW "A1;BM128,96;U25"
```

Run the program. The line is now horizontal.

Just Shootin' Blanks

If you want the next line you draw to be a "blank" or an invisible line, include the B option.

For example, let's say you are drawing letters of the alphabet and are ready for the letter C, which is nothing but a square with the right side blank. Change Line 30 as follows so the program generates such a figure:

```
30 DRAW "BM128,96;U25;R25 ;B  ;D25;L25"
```

Run the program. Remember, only the line immediately following the B is blank.

DO-IT-YOURSELF PROGRAM 21-3

Print your name on the screen using DRAW. This means you'll have to stay in the graphics screen. Sure, it would be easier to write your name on the text screen, but you can't have "true" text and graphics at the same time.

What! More Options?

Another of DRAW's many features is N, the "no update" option. N tells the computer to return to its original (current) position after it draws the next line. To see this, change Line 30 to read:

```
30 DRAW "M128,96; N; U25; N; R25; N;
   D25; N; L25;"
```

Run the program. The computer draws a 25-point line straight up from (128,96). It then returns to (128,96), draws the next line, returns, draws the next, and so on. As a result, four lines radiate from the center of the screen, each in a different direction (up, right, down, and left).

DO-IT-YOURSELF PROGRAM 21-4

Using DRAW's N option (and CIRCLE), have the computer draw a pie that has 8 pieces. Once you've done that, cut out a piece of the pie and put it over to one side.

String Constants *v* String Variables

As stated earlier, the *string* following DRAW can be either a constant—as in the previous examples—or a variable.

To use a string variable, precede the DRAW statement with a program line that identifies the variable as a string; then substitute the string for the quoted material in DRAW. For example, add Line 25 and change Line 30 as follows:

```
25 A$="BM128,96;C8;U25;R25;D25;L25"
30 DRAW A$
```

Does that mean it's a drawstring?

A semicolon must always follow the dollar sign (even though the other semicolons are not necessary):

```
XA$;XX$;XC$
```

Run the program. The computer displays an orange box (25 x 25), the lower left corner of which is in the center of the screen.

Extended Color BASIC offers a variation on this, called the "execute" (X) action. While you execute a DRAW routine, the execute action lets you execute another DRAW string, then return to and complete the first operation. To do this, leave Line 25 as is so that it defines A$; then change Line 30. The two lines read:

```
25 A$="BM128,96;C8;U25;R25;D25;L25"
30 DRAW "BM95,50;U25;R25; XA$; D25;L25"
```

Run the program. The computer starts drawing at (95,50) a line that extends up (U25) and then right (R25). It then executes A$ so that it draws a 25 x 25 square, starting at (128,96). After executing A$, it returns to the original (current) string and completes its execution (D25,L25).

DO-IT-YOURSELF PROGRAM 21-5

Do-It-Yourself Program 21-3 shows that you can simulate text (letters) on the graphics screen by drawing the letters. Use DRAW to create all 26 letters of the alphabet. Store the DRAW commands in strings. Then use the "execute" (X) action to arrange the letters into words.

DO-IT-YOURSELF PROGRAM 21-6

Do you still have your house? If so, load the program again and use DRAW to make the front door open and close.

Learned in Chapter 21

BASIC WORDS	CONCEPTS
DRAW	Drawing visible lines Drawing invisible (blank) lines Scaling figures to size Coloring lines Returning the draw to its original position Using string variables to draw Executing a second draw in the middle of the first

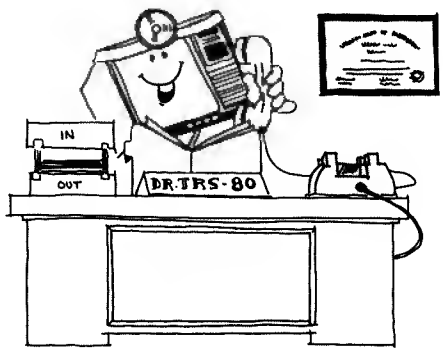
Notes

CHAPTER 22

GET AND PUT: THE DISPLAY WENT THAT ARRAY

In previous chapters, you've learned a few ways to move figures from one screen to another, but none is very efficient. Have no fear; there is a better array (groan). It has to do with GET and PUT.

Using these statements, you can "get" a rectangular area from the screen, store its contents in an "array" (an area of memory), and then "put" it back anywhere you want on the screen. This is the best method for simulating motion.



A who? A what? Arrays are covered in Part III, later in this manual.

We use the term "rectangle" to refer to the area that contains the graphic display. Of course, you can't actually see the rectangle. You'll have to visualize it. Here's an illustration to help you:

The formats for GET and PUT are:

GET *h1,v1-h2,v2,array,* G gets a rectangle from the current graphics screen and stores it in an array

h1,v1 is the rectangle's upper-left corner.
h2,v2 is the rectangle's lower-right corner.
array is an area in memory that stores the rectangle.
G stores the array in full graphic detail. It is required when using high resolution (Mode 4 or Mode 3 with colors) or when using the PUT *action* parameters. Otherwise, garbage appears on your screen.

PUT *h1,v1-h2,v2,array,action* puts a rectangle, stored in an array, on the current graphics screen

h1,v1 is the rectangle's upper-left corner.
h2,v2 is the rectangle's lower-right corner.
array is an area in memory where the rectangle is stored.
action (shown on Table 22-1) tells the computer what to do with the points stored in the rectangle.

Note: Be sure the computer is in the same PMODE for GET as it is for PUT. Otherwise, you may not "put" what you "got."

Type and run this program to see how GET and PUT work:

```
5 PCLEAR 4
10 PMODE 3,1
15 PCLS
20 SCREEN 1,1
25 DIM V(20,20)
30 CIRCLE (20,20),10
35 GET (10,10)-(30,30),V
40 PCLS
42 FOR DLAY = 1 TO 300: NEXT DLAY
45 PUT (110,110)-(130,130),V
50 FOR DLAY = 1 TO 300: NEXT DLAY
60 GOTO 60
```

The program draws a circle on one part of the screen and then moves it to another. To do this, the computer:

1. Creates an array named V in memory (Line 25). Array V is big enough to store a 20 X 20 rectangle.
2. Draws a circle on the screen (Line 30).
3. Gets a 20 X 20 rectangle containing the circle and stores it in the Array V (Line 35).
4. Clears the screen (Line 40).
5. Puts the 20 X 20 rectangle (stored in Array V) back on the screen.

Storing the Rectangle

As you can see from the above program, GET and PUT use an array to store the rectangle. So, before you use GET or PUT, you need to create this array.

The DIM statement lets you do this.

DIM *array(length, width)* creates an array for storing a rectangle the size of *length* X *width* points

Note: DIM should be one of the first lines in your program (after CLEAR and PCLEAR, if you use them).

How large a rectangle you can store in an array depends on how much memory you have. Each point, when stored in an array, consumes 5 bytes of memory. In a 16K RAM system, you can store no more than 1400 points in an array. If your program is long, you may have to use a smaller array.

How large does the array need to be? This depends on how large a rectangle you want to "get" or "put":

Width = *h2 - h1*
Length = *v2 - v1*

For example, the above program's GET statement uses (10,10) and (30,30) to specify a rectangle. Thus, the rectangle is 20 X 20: It has a *width* and *length* of 20. The PUT statement uses the same size rectangle: 20 X 20.

Put Not What You See

You've now put a rectangle on the screen one way—with the PSET *action*. (When you don't specify another *action*, the computer uses PSET.) There's more than one way, though, to put rectangles on the screen.

To see how the other *actions* work, start by running this program. It puts 15 rectangles on the screen with the PSET action.

```
5 PCLEAR 4
10 DIM V (30,30)
15 PMODE 2,1
20 PCLS
25 SCREEN 1,1
30 CIRCLE (128,96),30
35 PAINT (128,95),2,4
40 PAINT (128,97),3,4
45 GET (98,81)-(128,111),V,G
50 PCLS
55 FOR I = 150 TO 1 STEP -10
60 PUT (I,81-I/5)-(I+60,111-I/5),V,PSET
65 NEXT I
70 GOTO 70
```

If the computer puts garbage on your screen, perhaps you have omitted the G option with GET.

PSET sets and resets each point as it is in the array rectangle. Each rectangle it puts on the screen is the same as the one stored in the array.

Now change Line 60 in various ways to try other actions. First, try PRESET.

```
60 PUT (I,81-I/5)-(I+60,111-I/5),V,PRESET
```

PRESET sets and resets the reverse of each point in the array rectangle. Each rectangle it puts on the screen is the reverse of the one stored in the array.

Try the OR action:

```
60 PUT (I,81-I/5)-(I+60,111-I/5),V,OR
```

OR sets each point that's either (1) set in the array rectangle *or* (2) already set in the position where it's putting the screen rectangle. Each rectangle it puts on the screen has all points set that are stored in the array plus what is currently on the screen.

For a strange effect, try the NOT action:

```
60 PUT (I,81-I/5)-(I+60,111-I/5),V,NOT
```

NOT sets and resets the reverse of what's on the screen. (NOT doesn't care what's stored in the array.) Each rectangle it puts on the screen is the reverse of the previous one.

Try the AND option with this program, and you won't see anything:

```
60 PUT (I,81-I/5)-(I+60,111-I/5),V,AND
```

AND sets each point that (1) is set in the array *and* (2) is already set on the screen in the position where it's putting the rectangle. Any points that don't meet both of those conditions are reset. In this case, each rectangle AND puts on the screen has all points reset—you see nothing.

This is a summary of each action:

Option	Function
PSET	Sets each point that is set in the array.
PRESET	Resets each point that is set in the array; sets each point that is reset in the array.
AND	Compares each point in the array rectangle with the screen rectangle. If both are set, the computer sets the screen point; if not, it resets the screen point.
OR	Compares each point in the array rectangle to the screen rectangle. If either is set, the computer sets the screen point.
NOT	Reverses the state of each point in the screen rectangle regardless of the array rectangle's contents.

DO-IT-YOURSELF PROGRAM 22-1

Use GET and PUT to send a spaceship up the screen and across its "outer limits." You might want to add a few asteroids and aliens to make the voyage more exciting!

Learned in Chapter 22

BASIC WORDS	CONCEPTS
GET	Storing a screen display in an array
PUT	Returning the display to the screen in either the same or a different position
	Determining the state of the returned points of the display

Notes

CHAPTER 23

A NEW KIND OF POINT

As you recall from the SCREEN and PMODE chapters, your computer has two kinds of video memory—text and graphics. And it uses these two memories to create two kinds of screens—text and graphics.

All the extended graphics statements (such as LINE, CIRCLE, PPOINT, and PMODE) create graphics screens using the massive power of graphics memory. This lets you draw exciting, high-resolution, and fast-moving images.

There are two kinds of images, though, that you can't produce on a graphics screen:

An image that uses all 9 colors (You can use no more than 4 colors on a graphics screen.)

An image that uses text, as well as pictures (You cannot print text on a graphics screen.)

To produce these kinds of images, you need to draw pictures on a text screen. Extended Color BASIC has 3 statements you can use for this purpose:

SET—sets a dot on your text screen

RESET—resets a dot on your text screen

POINT—tells what color a dot is on your text screen.

If these statements remind you of PSET, PRESET, and PPOINT, that's no accident. SET, RESET, and POINT perform the function on the text screen as PSET, PRESET, and PPOINT perform on the graphics screen.

The analogy ends there, though. There are no text screen equivalents to such powerful statements as DRAW, PAINT, and PMODE. On a text screen, you can draw only one dot at a time.

First make your screen black:

```
10 CLS(0)
```

Now set a dot—a blue one—on the top-left corner of your text screen. Type and run this program:

```
20  SET(0,0,3)
30  GOTO 30
```

Set another dot—a buff one—on the bottom-right corner of your screen.

```
20  SET(63,31,5)
```

As you may see, you do not use the 256 X 192 graphics grid to set dots on your text screen. Instead, you use a 31 X 63 grid called the SET/RESET grid (shown in the back of this book).

Setting Two Dots

To set two dots on a text screen, you need to plan. To find out why, run a few programs. First, type and run this:

```
10  CLS(0)
20  SET(32,14,3)
30  SET(33,14,3)
40  GOTO 40
```

You should now have two blue dots—side by side—in the middle of your screen. Change the color of the right dot so you'll have one blue and one red dot. Type:

```
30  SET(33,14,4)
```

Run the program again. This time, *both dots are red.*

Look again at the SET/RESET grid. Notice that the darker lines group the dots into "blocks." Each block contains 4 dots. For instance, the block in the middle of the grid contains these 4 dots:

	Horizontal	Vertical
Position	32	14
Position	33	14
Position	32	15
Position	33	15

Each dot within a block must either be:

the same color
or

black

The above program asks the computer to set two different-colored dots (red and blue) within the same block. Since the computer can't set them in different colors, it sets them both the second color: red.

When you reset a dot on the text screen, the computer makes the dot black.

Type and run this program:

```
30  SET(34,14,4)
```

Since the dot in Position 34, 14 is in a different block, the computer can set the two dots in different colors.

The Computer's Face

Drawing pictures on the text screen will seem primitive to you after using statements such as CIRCLE, DRAW, and PAINT. But if you want pictures and text, you can use the following program as a guide.

Run this program, and you see computer's face and text on the same screen. SET and RESET draw the picture (using the SET/RESET grid) and PRINT @ prints the text (using the PRINT @ grid).

```
5 CLS(0)
7 PRINT @ 397, "HELLO";
10 FOR H = 15 TO 48
20 SET(H,5,5)
30 SET(H,20,5)
40 NEXT H
```

```
50 FOR V = 5 TO 20
60 SET(15,V,5)
70 SET(48,V,5)
80 NEXT V

90 SET(32,13,8)

100 FOR H = 28 to 36
110 SET(H,16,4)
120 NEXT H

130 SET(25,10,3)
140 SET(38,10,3)

150 RESET(38,10)
160 GOTO 140
```

Notice we've changed Line 50—the GOTO line.

Notice that this program is able to draw *5* colors on one screen—and could actually draw *all 9 colors.*

These are the formats of SET, RESET, and POINT:

SET *h,v,c* sets a point on the text screen

h is the horizontal coordinate (0-63)
v is the vertical coordinate (0-31)
c is the color code (0-8)

RESET *h,v* resets a point on the text screen

h is the horizontal coordinate (0-63)
v is the vertical coordinate (0-31)

POINT *h,v* tells what color a point is on the text screen

h is the horizontal coordinate (0-63)
v is the vertical coordinate (0-31)

If You Have the Joysticks . . .

If you have joysticks, connect them now by plugging them into the back of your computer. They fit in only the correct slots, so don't worry about plugging them into the wrong places.

Now run this short program to see how joysticks work:

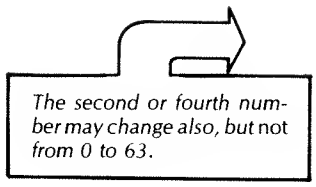
```
10  CLS
20  PRINT @ 0, JOYSTK(0);
30  PRINT @ 5, JOYSTK(1);
40  PRINT @ 10, JOYSTK(2);
50  PRINT @ 15, JOYSTK(3);
60  GOTO 20
```

Be sure to type the semicolons at the ends of Lines 20, 30, 40, and 50.

See the 4 numbers on your screen? They're the horizontal and vertical positions of the 2 joysticks' "floating switches."

Grasp the right joystick's floating switch. (The joystick connected to the RIGHT JOYSTICK jack on the back of the computer.) Keeping it in the center, move it from left to right. The first number on the screen changes from 0 to 63, going through all the intervening numbers.

Move the left joystick's floating switch from left to right. The third number on the screen changes.



The second or fourth number may change also, but not from 0 to 63.

Now move the floating switches up and down, keeping them in the center. Moving the *right* joystick up and down changes the *second* number from 0 to 63. Moving the *left* joystick up and down changes the *fourth* number from 0 to 63.

This is how the computer reads the joysticks' positions:

JOYSTK(0) and JOYSTK(1) read the *right* joystick's positions:

JOYSTK(0) reads the horizontal (left to right) coordinate.

JOYSTK(1) reads the vertical (up and down) coordinate.

JOYSTK(2) and JOYSTK(3) read the *left* joystick's positions:

JOYSTK(2) reads the horizontal coordinate.

JOYSTK(3) reads the vertical coordinate.

Whenever you read any of the joysticks, you must read JOYSTK(0). To find out for yourself, delete Line 50 and run the program. It works almost the same, except it doesn't read JOYSTK(3) — the vertical position of your left joystick.

Delete Line 20 and change Line 60:

```
60  GOTO 30
```

Run the program. Move all the switches around. This time the program doesn't work at all. The computer won't read any coordinates unless you first have it read JOYSTK(0). Type these lines and run the program:

```
20  A = JOYSTK(0)
60  GOTO 20
```

Although the computer's not printing JOYSTK(0)'s coordinates, it's still reading them. Because of this, it's able to read the other joystick coordinates. Whenever you want to read JOYSTK(1), JOYSTK(2), or JOYSTK(3), you first need to read JOYSTK(0).

Painting with Joysticks

Type and run this program:

```
10  CLS(0)
20  H = JOYSTK(0)
30  V = JOYSTK(1)
40  IF V > 31 THEN V = V - 32
80  SET(H,V,3)
90  GOTO 20
```

This program uses joysticks with text screen pictures. You can just as easily use the joysticks with graphics screen pictures.

Use the revolving switch of your right joystick to paint a picture. (Move the switch slowly so that the computer has time to read its coordinates.)

Line 20 reads H—the horizontal position of your right joystick. This can be a number in the range 0 to 63.

Line 30 reads V—its vertical position. This also can be a number in the range 0 to 63. Since the highest vertical position on your screen is 31, Line 40 is necessary: It makes V always equal a number in the range 0 to 31.

Line 80 sets a blue dot at H and V.

Line 90 goes back to get the next horizontal and vertical positions of your joysticks.

This uses only the right joystick. Perhaps you could use the left one for color. Add these lines and run the program:

```
50  C = JOYSTK(2)
60  IF C < 31 THEN C = 3
70  IF C > = 31  THEN C = 4
80  SET(H,V,C)
```

Move your left joystick to the right, and the computer makes C equal to 3; the dots it sets are red. Move it to the left, and the computer makes C equal to 4; the dots it sets are blue.

Want to use your joystick buttons? Add these lines to the program:

```
100  P = PEEK(65280)
110  PRINT P
120  GOTO 100
```

Now type:

```
RUN 100 (ENTER)
```

This tells the computer to run the program starting at Line 100. Your computer should be printing either 255 or 127 over and over.

PEEK tells the computer to look at a certain spot in its memory to see what number's there. Line 100 looks at the number in Position 65280. As long as you're not pressing either of the buttons, this spot contains the number 255 or 127.

If you press the buttons when you're not running the program, you'll see @ABCDEFG or HIJKLMNO.

Press the right button. When you press it, this memory location contains either the number 126 or 254.

Press the left button. This makes this memory location contain either the number 125 or 253.

Using this information, you can make the computer do whatever you want when you press one of the buttons. We'll make it go back to Line 10 and CLS(0) (clear the screen to black) when you press the right button. Change Lines 110 and 120:

Some joysticks will not read six "blocks" in each of the four corners of your screen.

```
110  IF P = 126 THEN 10
120  IF P = 254 THEN 10
```

Delete Line 90 and add this line:

```
130  GOTO 20
```

Run the program and start "painting." Press the right button when you want to clear the screen and start again.

Learned in Chapter 23

BASIC WORDS

SET
RESET
JOYSTK
PEEK

Notes

CHAPTER 24

PLAY IT AGAIN, TRS-80

So you think your computer is a good artist, huh? Well, you haven't heard anything yet! Wait until you find out about its musical talents! Ready? Then let's get down to work and PLAY.

Your computer's PLAY function allows you not only to play music, but to compose it, as well.

Note: PLAY, of course, is not a graphics function. Therefore, you needn't preface your programs with PMODE, PCLS, or SCREEN.

Listen Carefully . . .

Here is the syntax for PLAY:

PLAY *music* plays the value of music, a string expression including the following:

note (a letter from "A" to "G" or a number from 1 to 12).
octave (O followed by a number from 1 to 5). If you omit the *octave*, the computer uses Octave 2.
note-length (L followed by a numeral from 1 to 255). If you omit the *note-length*, the computer uses the current length.
tempo (T followed by a number from 1 to 255). If you omit the *tempo*, the computer uses T2.
volume (V followed by a number from 1 to 31). If you omit the *volume*, the computer uses V15.
pause-length (P followed by a number from 1 to 255).
substrings. Precede substrings with an X and follow them with a semicolon. Example: XA$;

Let's Compare Notes
(NOTE)

Obviously, you can't have music without notes. PLAY gives two ways to specify the precise note you need.

The first—and probably easier—way to play the note you want is to enter one of the standard musical notes—A, B, C, D, E, F or G. To indicate a sharp note, follow the note with a plus sign (+) or with the pound sign (#). To indicate a flat, follow it with a minus sign (−).

For example, A represents A natural; A# is A sharp; and A− is A flat. Type the following to see (hear?) what we mean:

```
PLAY "A" (ENTER)
```

To hear the change that a sharp and a flat can make, enter these lines:

```
PLAY "A;A#" (ENTER)
PLAY "A-;A;A#;A;A-" (ENTER)
```

You can do the same with all seven notes (A-G) on the scale, except B and C. Since B# = C, you must use C. Likewise, since C− = B, you must use B.

A New "Note"-ation

Another way to specify a musical note is to use a number between 1 and 12, prefaced by the letter N. (If you omit N, the number alone indicates the note.)

The numbers 1 through 12 represent every note on the musical scale, including all sharps and flats. This is a more concise notation, although it is more difficult to read if you already know the standard notation.

Note: Since PLAY does not recognize the notation B# or C−, use the numbers 1 and 12, respectively, or substitute C for B# and B for C−.

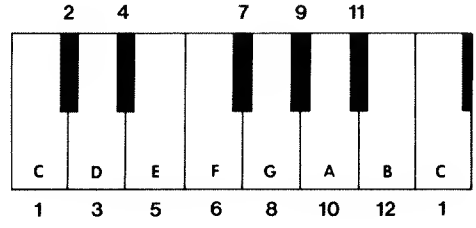
To hear the full 12-tone scale, run the "Scale" program, which follows.

```
5 CLS
10 FOR N = 1 to 12 'N = NOTE
15 PRINT "NOTE#"; N
20 PLAY STR$(N)
30 NEXT N
```

Add a delay in the program so you can compare the numbers to the notes as the scale goes up from 1 to 12 (C to B).

```
25 FOR I = 1 TO 500: NEXT I
```

Musical Note/Number Table	
Number	**Note**
1	C
2	C#/D–
3	D
4	E–/D#
5	E/F–
6	F/E#
7	F#/G–
8	G
9	G#/A–
10	A
11	A#/B–
12	B



DO-IT-YOURSELF PROGRAM 23-1

Modify the "Scale" program so it goes down instead of up.

Whole Notes, Half Notes, Quarter Notes . . .
(NOTE-LENGTH)

Because the "Scale" program does not specify note-length, the computer automatically uses quarter notes, the initial "current value."

Did you time the notes to be sure they are four times as long? It's not necessary; the computer's internal clock did it for you.

To choose the note-length, use L followed by a number from 1 to 255. The number 1, for instance, denotes a whole note, 2 a half note, 4 a quarter note, 8 an eighth note, 16 a sixteenth note, and so on.

In fact, you can use any number from 1 to 255. (Who ever heard of a 1/15th note?)

Vary the note-lengths to produce a drum roll. Type:

```
PLAY "L2;A;L4;A;A;L2;A;A" ENTER
```

Lnumber	**Note-Length**	**Note**
L1	Whole note	𝅝
L2	Half note	𝅗𝅥
L3	Dotted quarter note	𝅗𝅥.
L4	Quarter note	♩
L8	Eighth note	♪
L16	1/16 note	𝅘𝅥𝅯
L32	1/32 note	𝅘𝅥𝅰
L64	1/64 note	𝅘𝅥𝅱
.		
.		
.		
L255	1/255 note	

L2 indicates a half note; L4 a quarter note, so we played as follows: "half, quarter, quarter, half, half."

```
PLAY "L1; A;A#;A-" (ENTER)
```

Notice that you needn't repeat the L option for each note. PLAY uses the current note value until you enter another L command to tell it otherwise.

In fact, most PLAY options discussed in the rest of this chapter use a "current" value until you change them.

Just for fun, try playing three 1/255 notes on A:

```
PLAY "L255;A;A+;A-" (ENTER)
```

Now that's staccato.

Love That Dotted Note

If you read music, you already know about "dotted notes." The dot tells you to increase the length of the note by one half its normal value. For example, a dotted quarter note is equal to a "3/8" note.

You can play such a note by adding a period (.) or a series of periods (. . .) to the L*number*. Each period increases the note-length by 1/2 its normal value. For example:

L4. = 1/4 + 1/8 = a 3/8 note

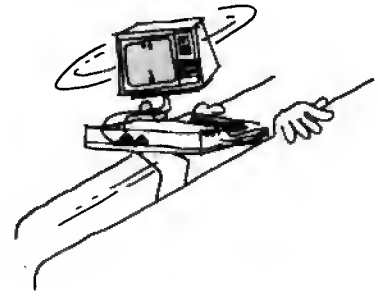
Try this:

```
PLAY "L4.;A;L8;C;L4.;E;L8;C;E;C;E;C;L4;A"
(ENTER)
```

Let's Go Up (or Down) an Octave or Two (OCTAVE)

Our single octave (Octave 2) sounds fine, but, variety being the spice of music as well as Irish stew, it gets a little boring when played over and over (like a piano with only 12 keys).

We bet you've heard of "turning down the stereo" but not "turning down the computer"!



To change octaves, use the letter O followed by a number in the range 1 to 5. (Any number out of this range results in an illegal function call error.)

If you don't specify the octave, the computer automatically uses Octave 2, which includes middle-C. Let's try to play a simple C scale:

```
PLAY "CDEFGABAGFEDCBA" (ENTER)
```

What happened? G is the highest note in Octave 2, so when the computer reached A, it started over at the beginning of the octave. To get out into Octave 3, try this:

```
PLAY "CDEFG;03;ABA02;FEDCBA" (ENTER)
```

Play It Again—Louder!
(VOLUME)

Sure, you can adjust the volume of your music by using the TV volume control, but who wants to sit by the set all of the time? Especially when the computer can adjust the volume for you.

Your computer does this with the V (volume) feature. All you need to do is use V followed by a numeral between 0 and 31. If you don't specify the value of V, your computer automatically uses V15.

The computer uses the current V value until you change it.

Adjust the volume on your TV to a normal setting and run this short program:

```
5 CLS
10 PLAY "V5;A; V10;A; V15;A; V20;A; V25;A;
   V30;A"
20 GOTO 10
```

Getting a headache? Press (BREAK) to get out of the loop.

A Moment of Silence, Please
(PAUSE)

Maybe that last little program would be easier to listen to if all the notes weren't played together. Use the P (pause) feature for a few moments of silence between the notes and see if they sound better.

To put a pause between notes, use P followed by a number from 1 to 255. Pause-lengths correspond to note-lengths with one important difference. You can't use dots (periods) with P. To compensate, just type a series of pauses. For example, to get a 3/8 pause, type P4P8.

Change Line 10 in the last program to read:

```
10 PLAY "V5;A; P2; V10;A; P2; V15;A;
   P2; V20;A; P2; V25;A; P2; V30;A; P2"
```

Actually, a half note pause (P2) between all those As doesn't make them sound much better, but you should get the idea of how P works.

We've left spaces between each volume/note combination so you can read the line without difficulty. The spaces are not required.

It's Time to Pick Up the Tempo
(TEMPO)

Right now the test program looks like this:

```
5 CLS
10 PLAY "V5;A;P2; V10;A;P2; V15;A;P2;
   V20;A;P2; V25;A;P2; V30;A;P2"
20 GOTO 10
```

It's passable, if not pleasurable, but the tempo (speed) is a little slow. You can increase or decrease the tempo with T and a number from 1 to 255. If you don't specify a tempo, your computer automatically uses T2. Start by slowing down the tempo of the program:

```
10 PLAY "T1; V5;A;P2; V10;A;P2; V15;A;P2;
   V20;A;P2; V25;A;P2; V30;A;P2"
```

A tempo that slow is almost enough to keep you awake—almost.

Speed it up by changing T1 to T15. Now that's more like it.

How about speeding it to the maximum, 255, and running the program. That didn't take long, did it?

Executing the Substring (X)

Remember DRAW's execute (X) option? PLAY has a similar feature that lets you execute a substring, then return to the original string and complete it.

If you use machine-language to generate the music and "Tn" to slow the tempo, would your computer be a Slow-POKE?

The execute function takes the following form:

```
XA$;
```

A$ contains a sequence of normal play commands and functions. X tells the computer to PLAY A$.

Rearrange the demonstration program so that it executes a substring:

```
5 CLS
10 A$ = "A;A#;A-"
20 B$ = "O5;XA$;"
30 C$ = "O1;XA$;XB$;"
40 PLAY C$
```

Run the program and follow its execution.

Note: Whenever you use the execute function, a semicolon (;) must follow the dollar sign ($). In this example, you can delete all semicolons except those following the dollar sign.

One Further Note . . . (+,−,<,>)

No, we're not going to spring a new note, like H or J, on you. We just have one final way you can use some of PLAY's options. With O (octave), V (volume), T (tempo), and L (note-length), you can use one of the following suffixes instead of adding a numeral:

Suffix	Purpose
+	Adds 1 to the current value.
−	Subtracts 1 from the current value.
>	Multiplies the current value by 2.
<	Divides the current value by 2.

Use the sample program to learn about these features.

```
5 CLS
10 PLAY "T2"
```

```
20 PLAY "A;A#;A-"
30 GOTO 20
```

Notice that Line 10 sets the tempo. Run the program once just to get an ear for it. Nothing's changed; it's the same as always. Now insert T in Line 20.

```
20 PLAY "T+; A;A#;A-"
```

Run the program. The plus sign automatically increases the value of T by 1 each time Line 20 is played. From a slow start you can really begin to fly! Did you hear it shift gears somewhere around T100?

Now reduce the tempo, using a minus sign (−):

```
5 CLS
10 PLAY "T255"
20 PLAY "T-; A;A#;A-"
30 GOTO 20
```

After a fast start, the computer finally manages to slow the tempo down to 1—one step at a time.

Isn't multiplication faster than addition? In Line 10, reset the tempo to 2, change T in Line 20 to T>, and let it rip.

```
10 PLAY "T2"
20 PLAY "T>; A;A#;A-"
```

You started out with T2, right? The computer multipled that value by 2 to 4, 4 x 2 to 8, 8 x 2 to 16, and so on until it reached 255.

You can slow the tempo down just as quickly by dividing the current tempo by 2 using "<."

```
10 PLAY "T255"
20 PLAY "T<; A;A#;A-"
```

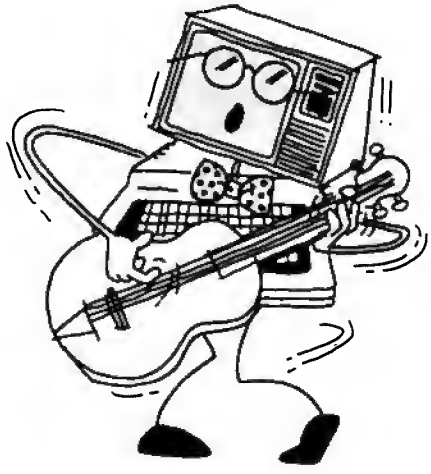
Remember, you can do the same thing with L, V, and O to change the note-length, the volume, and the octave.

Here is the formula that you can use to calculate the note-length: note-length + (note-length * number of dots)/2

Haven't you had days like that? You start at 255 and by the end of the day, you're hitting on one cylinder.

Roll Over, Beethoven

After all the hard work you've done lately, you deserve to be serenaded.



Are you familiar with all the PLAY functions? If so, watch them at work in the following program and see if you can name this tune!

```
5 CLS
100 A$ = "T5;C;E;F;L1;G;P4;L4;C;E;F;L1;G"
105 B$ = "P4;L4;C;E;F;L2;G;E;C;E;L1;D"
110 C$ = "P8;L4;E;E;D;L2.;C;L4;C;L2;E"
115 D$ = "L4;G;G;G;L1;F;L4;E;F"
120 E$ = "L2;G;E;L4;C;L8;D;D+;D;E;G;L4;A;L1
        ;03;C"
125 X$ = "XA$;XB$;XC$;XD$;XE$;"
130 PLAY X$
```

Do you recognize that song? Dress it up a bit by adding these lines:

```
10 PRINT @ 96, STRING$ (32,"*")
20 PRINT @ 167, "WHEN THE SAINTS"
30 PRINT @ 232, "GO MARCHING IN"
35 PRINT @ 288, STRING$ (32,"*")
40 FOR X = 1 TO 500: NEXT X
45 CLS
50 PRINT @ 128, "OH WHEN THE SAINTS"
55 PRINT @ 169, "OH WHEN THE SAINTS"
60 PRINT @ 192, "OH WHEN THE SAINTS
   GO MARCHIN IN"
65 PRINT @ 224, "YES I WANT TO BE IN THAT
   NUMBER"
70 PRINT @ 256, "WHEN THE SAINTS GO MARCHIN
   IN"
```

We dropped the "G" from MARCHING in Lines 60 and 70 so the lines can fit on the screen.

Run the program now and sing along with TRS-80. What? You liked it so much you want to hear it again. Okay, add these lines:

```
150 CLS
160 PRINT @ 130, "PLAY IT AGAIN, TRS-80"
165 FOR X = 1 TO 500: NEXT X
170 CLS
175 PRINT @ 233, "I'D BE GLAD TO"
180 FOR I = 1 TO 500: NEXT I
185 GOTO 5
```

If you use PCLS3 to clear the graphics screen and then make the computer play a sad song, does that mean it's singing the blues?

DO-IT-YOURSELF PROGRAM 24-2

Our rendition of "Saints" sounds fine, but it isn't true New Orleans style. Jazz it up to suit your own musical tastes. Try changing octaves or adding a few sharps or flats.

DO-IT-YOURSELF PROGRAM 24-3

Try some musical arrangements of your own. We've included several in the Sample Programs at the back of the book.

Learned in Chapter 24

BASIC WORDS	CONCEPTS
PLAY	Generating musical notes, including dotted notes Determining note-length Changing octaves Adjusting the volume Pausing between notes Changing the tempo Executing substrings Using suffixes to give values relative to the current value

Notes

SECTION III

GETTING DOWN TO BUSINESS

This section deals with information you want to manage. For example, you may want to manage:

Checkbook receipts

Shopping items

Tax records

Inventory

Addresses

Records, books, or tape collections

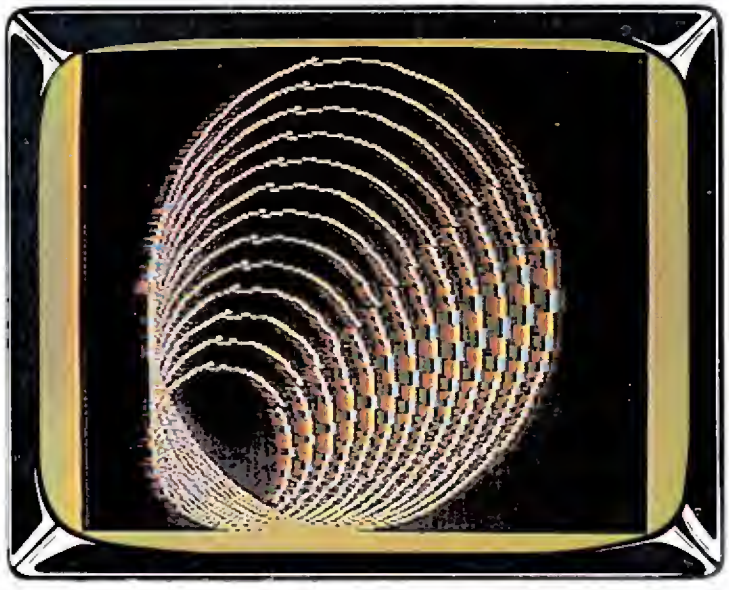
In this section, you'll learn how to store, update, sort, and analyze information to fit your own needs.

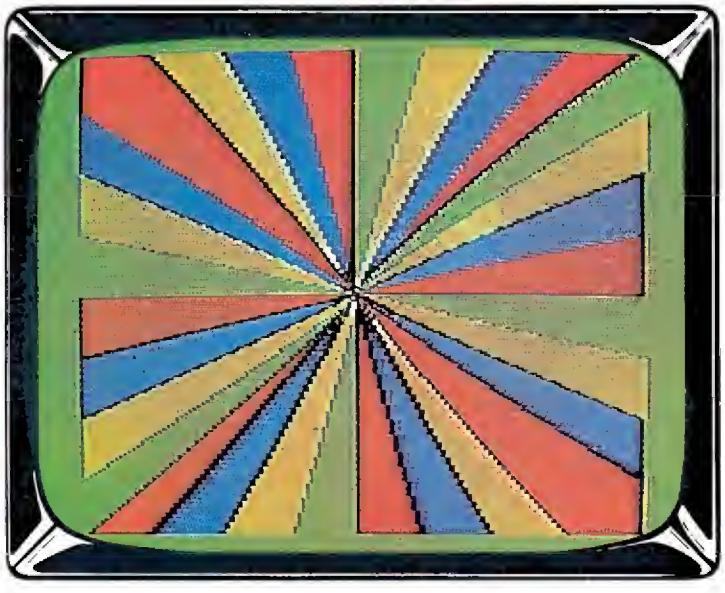
THE REAL THING

A special section showing displays created by programs in this book.

Spiral

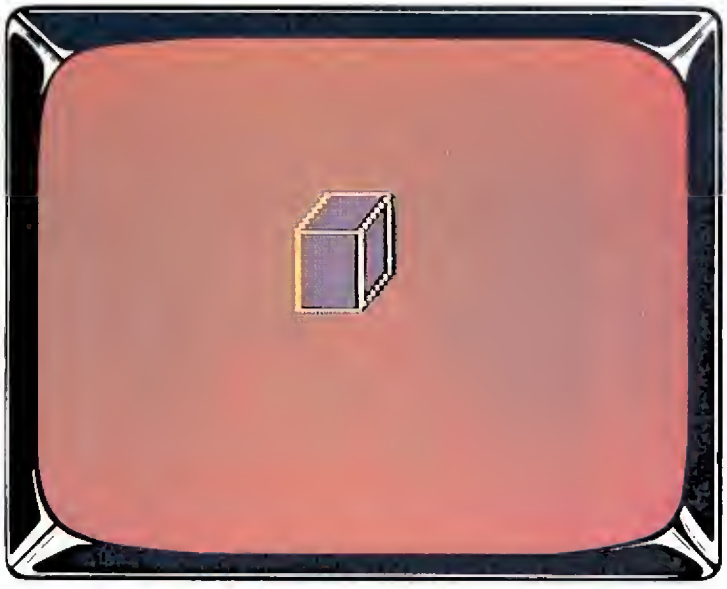
There's a tunnel at the end of the tunnel. When you assign variables to CIRCLE, it's possible to create a spiral. This is one way you can simulate smoke coming from the chimney of your house (see DO-IT-YOURSELF PROGRAM 19-4).





Fantastic!

DO-IT-YOURSELF PROGRAM 21-2 shows you how to cool off with an ice-cube. Another way is to turn on the fan and watch it spin. And if you let this program run for a while, that's exactly what happens. See Sample Program #19 for a listing of this program.

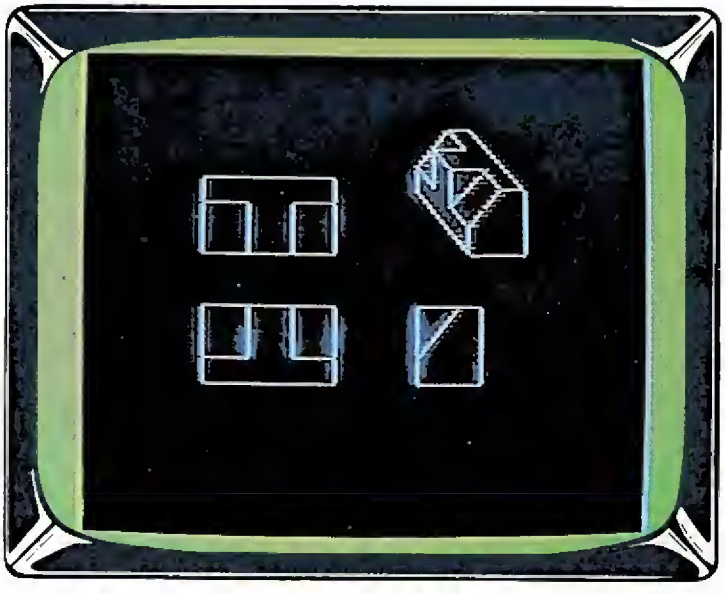


Box

This is a 2-step process. First the cube (created by DRAW and PAINT) appears in its 3-dimensional form. After a short delay, the box unfolds so you can see all 6 of its sides. This uses DRAW, along with several LINE and PAINT statements. See Sample Program #8.

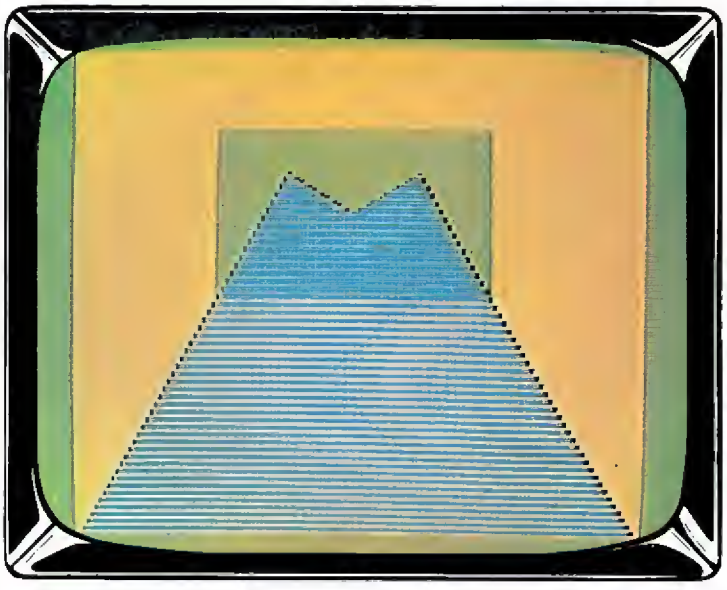
Projection Studies

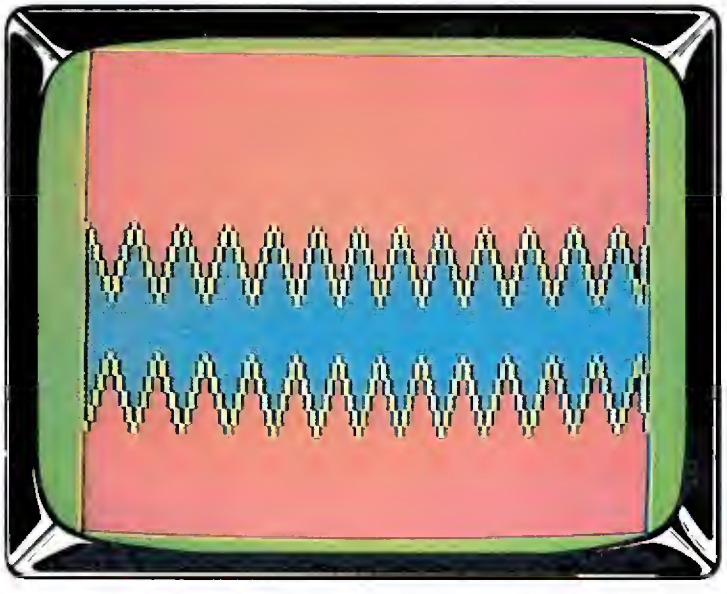
Starting at the upper left and going down, you can see different views (top, front, side, and oblique) of a "block." You can also scale the first three views up or down using DRAW's "Scale" feature. (Since the 45-degree oblique view contains three LINE statements, it can't be scaled.) See Sample Program #7.



In-Out

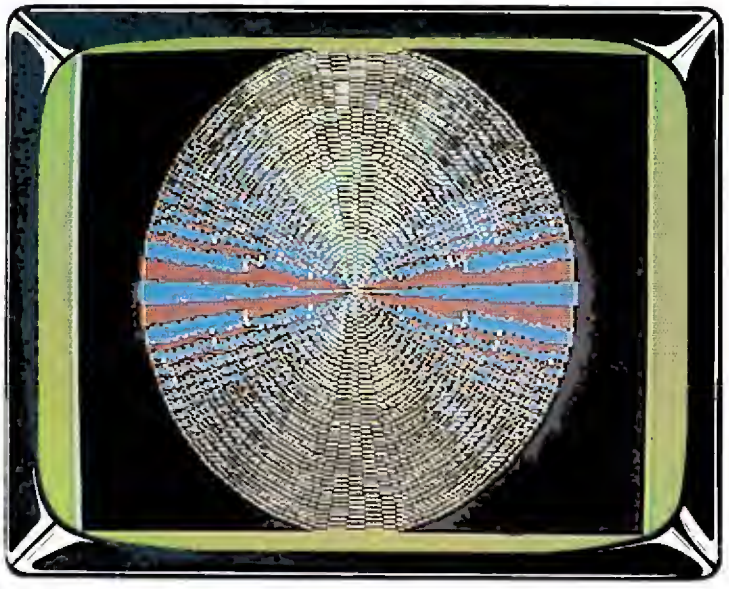
When you assign variables to a COLOR and a LINE statement, this is one thing the computer might do with it. Take a look at Sample Program #5 to see how easy this is.





Navaho Blanket

Actually, the size of this makes it more like a muffler instead of a blanket, but you should be able to finish "weaving" it. Basically, the program uses only a couple of LINE statements that increment at specified "steps" and a PAINT statement or two. Incidentally, this might help you with DO-IT-YOURSELF PROGRAM 15-3. Sample Program #12 gives you a complete program listing.

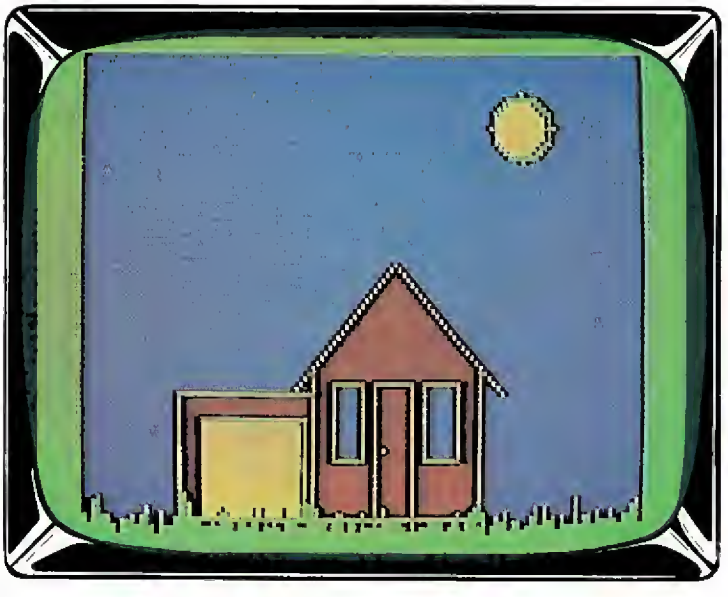


After the Boom Is Over . . .

These concentric circles (incrementing at STEP 2) are used at the end of the "Timebomb" program (Sample Program #18). Notice that when you use buff with high resolution, it appears to produce several colors, giving a metallic luster to the display.

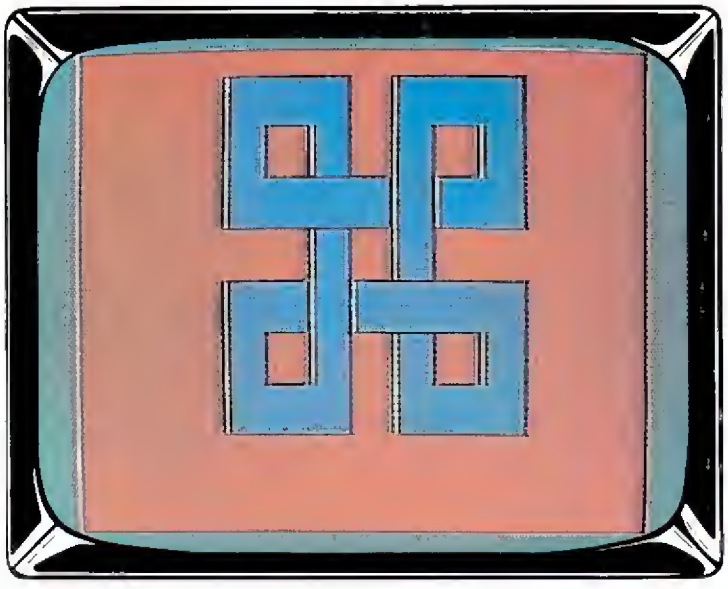
Home, Sweet Home

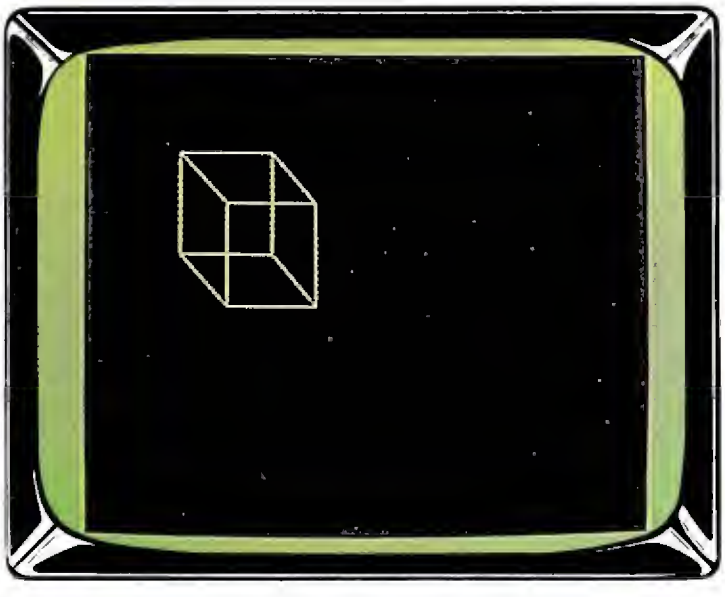
One of your exercises throughout this book is to "build" a house (see DO-IT-YOURSELF PROGRAMS 15-2 and 19-4). Here's one you might use as a model. In this instance, the garage door is up (using PAINT), the light is on, and the grass (generated by RND, DIM, and PSET) is growing.



Painted Lace

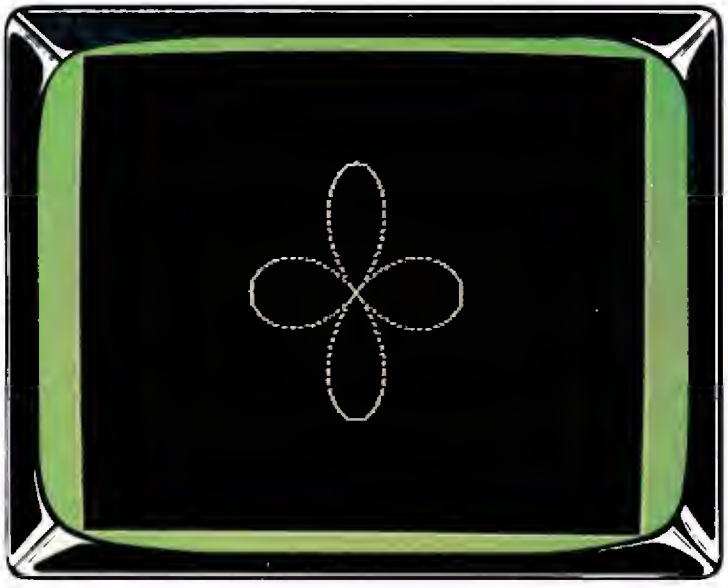
This program requires DRAW statements, a few buckets of PAINT, and a lot of patience. Look at Sample Program #13 and you'll see the way it's done.





Open and Closed Cubes

In DO-IT-YOURSELF PROGRAM 21-2 you drew the closed cube. Now "open" it.

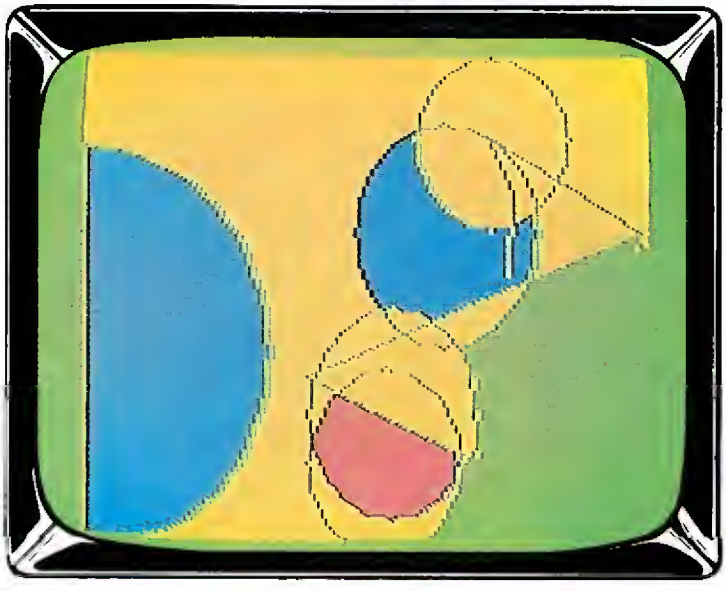


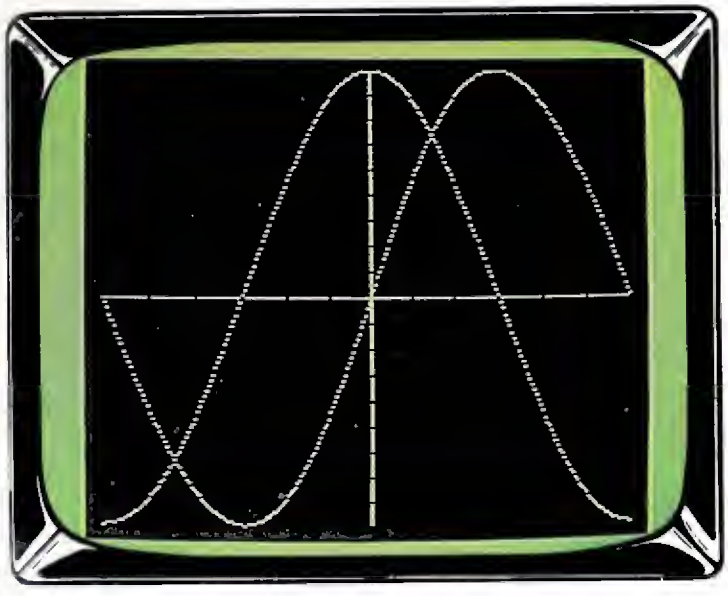
Rolling in the Clover

Sample Program #17 shows you how to create an eight-leaf clover. By changing the COS value in Line 35 to 2, you can generate a four-leaf clover. What happens if you change the COS value to 1? This program is a good illustration of PSET, SIN, and COS and a novel use of pi.

Random Graphics

Random graphics are generated when you assign random (RND) values to LINE, CIRCLE, COLOR, and PAINT and then let the computer take over. For a listing of this program, see Sample Program #11.





Riding the Waves

Here the computer uses PSET, SIN, and COS to draw sine/cosine waves and LINE to draw the H-V axes. Notice that each wave travels 360 degrees (from +180 to −180) and that the H-axis increments 30 degrees at each gradation. This is a good exercise in mapping (scaling down) a program to fit the TV screen. Sample Program #9 gives a complete listing of this program.

CHAPTER 25

TAPING

Your first and foremost task is to store your information permanently on cassette tape. This, of course, requires a tape recorder.

Ready to get organized? We'll start with your book collection. Here's a small list of books:

1. WORKING
2. CAT'S CRADLE
3. SMALL IS BEAUTIFUL
4. STEPPENWOLF

If you've read your introduction manual, you know how to save BASIC *programs* on tape. To save *information*, you need a program that follows these steps:

STEPS FOR STORING INFORMATION ON TAPE

1. *Open* communication to the tape recorder so that you can *output* (send out) information to a *file*.
2. *Output* all information to the tape recorder file.
3. *Close* communication to the tape recorder.

Start the program with this line:

```
10  OPEN "O", #-1, "BOOKS"
```

This "opens" communication to the tape recorder ("device #-1") so that you can "output" ("O") information. Whatever information you output, the computer stores on tape in a "file" named BOOKS.

A "file" is a collection of information—such as book titles—stored under one name.

Now output the information. Type:

```
15  CLS: PRINT "INPUT YOUR BOOKS--TYPE <XX>
    WHEN FINISHED"
20  INPUT "TITLE"; T$
30  PRINT #-1, T$
40  GOTO 15
```

Line 20 "prints" (outputs) your book titles—*not to the screen, but to device # - 1, the tape recorder.*

Then close communications. Type:

```
25  IF T$ = "XX" THEN 50
50  CLOSE #-1
```

The computer then closes communication to the tape recorder.

Add three more lines to the program:

```
1  CLS
2  PRINT "POSITION TAPE - PRESS PLAY AND
   RECORD"
4  INPUT "PRESS <ENTER> WHEN READY"; R$
```

The program should now look like this:

```
1  CLS
2  PRINT "POSITION TAPE--PRESS PLAY AND
   RECORD"
4  INPUT "PRESS <ENTER> WHEN READY"; R$
10  OPEN "O", #-1, "BOOKS"
15  CLS: PRINT "INPUT YOUR BOOKS - TYPE <XX>
    WHEN FINISHED"
20  INPUT "TITLE"; T$
25  IF T$ = "XX" THEN 50
30  PRINT #-1, T$
40  GOTO 15
50  CLOSE #-1
```

Prepare the recorder.

Connect the recorder. Your computer's introduction manual shows how.

Position a tape in the recorder, and, if necessary, rewind the tape so you'll have room for recording. (If you're using a non–Radio Shack tape, position it past the starting leader.)

Press the recorder's RECORD and PLAY buttons so that they are both down.

Then run the program. As soon as you press (ENTER), the cassette motor turns on: The computer is opening a "file" on tape and naming it BOOKS.

The computer clears the screen after each title.

The program then asks for titles. Type:

```
TITLE? WORKING
TITLE? CAT'S CRADLE
TITLE? SMALL IS BEAUTIFUL
TITLE? STEPPENWOLF
TITLE? XX
```

Each time you input a title, the computer prints it in a special place in memory reserved for the tape recorder. When you finish, the tape recorder motor turns on: The computer is printing all the titles to the recorder (Line 30) and then closing communication with the recorder (Line 50).

Your book titles are now all saved on tape in a file named BOOKS. To read them back into memory, use just about the same steps.

STEPS FOR INPUTTING INFORMATION FROM TAPE

1. *Open* communication to a tape recorder so that you can *input* information from a *file*.
2. Check to see if you're at the *end of the file*.
3. *Input* information from the tape recorder file.
4. Repeat Steps 2 and 3 until you reach the end of the file.
5. *Close* communication to the tape recorder.

To open communication, type:

```
60  CLS: PRINT "REWIND THE RECORDER AND
    PRESS PLAY"
70  INPUT "PRESS <ENTER> WHEN READY"; R$
80  OPEN "I", #-1, "BOOKS"
```

This opens communication to the tape recorder—this time, to input information from the BOOKS file.

To input information, add these lines:

```
90  INPUT #-1, B$
100  PRINT B$
```

Line 90 inputs the first book title (B$) from the BOOKS file stored on tape. (The variable name you choose makes no difference.) Line 100 displays this title on your screen.

To check for the end of the file and close the file, add these lines:

```
85  IF EOF (-1) THEN 120
110  GOTO 85
120  CLOSE #-1
```

Are you wondering what the -1 means? EOF returns a -1 when you reach the end of the file.

Line 85 says if you are at the end of this file (in this case, the BOOKS file), go to 120 and close communication with the tape recorder.

Note that EOF(-1) comes *before* the INPUT #-1 line. If it's *after* INPUT #-1, you'll get an IE error—"input past the end of the file."

List this last part of the program by typing LIST 60 - (ENTER). It should look like this:

```
60  CLS: PRINT "REWIND THE RECORDER AND
    PRESS PLAY"
70  INPUT "PRESS <ENTER> WHEN READY"; R$
80  OPEN "I", #-1, "BOOKS"
85  IF EOF (-1) THEN 120
90  INPUT #-1, B$
100  PRINT B$
110  GOTO 85
120  CLOSE #-1
```

Be sure to press only the PLAY button, Not RECORD. Also, be sure to rewind the tape.

If your computer becomes "hung up" communicating with the tape recorder, you can regain control by pressing the RESET button. It's on the back right-hand side of your keyboard. Then look for missing or mistyped lines in your program.

Now run this part of the program. Type:

```
RUN 60 (ENTER)
```

When you press (ENTER), the recorder's motor comes on while the computer inputs items from tape. When finished, it displays the four items on your screen.

An Electronic Card Catalog

Assume you need to change the program so it can also store the books' authors and subjects:

TITLE	AUTHOR	SUBJECT
Working	Studs Terkel	Sociology
Cat's Cradle	Kurt Vonnegut	Fiction
Small Is Beautiful	E. F. Schumacher	Economics
Steppenwolf	Hermann Hesse	Fiction

Start by changing the "output" part of the program (the first half). Type these lines:

```
26  INPUT "AUTHOR"; A$
28  INPUT "SUBJECT: S$
29  IF A$ = "XX" OR S$ = "XX" THEN 50
30  PRINT #-1, T$, A$, S$
```

Then change the "input" part of the program. Type these lines:

```
90  INPUT #-1, B$, A$, S$
100  PRINT "TITLE :" B$
102  PRINT "AUTHOR :" A$
104  PRINT "SUBJECT :" S$
```

Now take advantage of this organization. For example, have the program print a book list on any given subject. Add these lines:

```
130  CLS
140  INPUT "WHICH SUBJECT"; C$
150  PRINT "REWIND THE TAPE - PRESS PLAY"
160  INPUT "PRESS <ENTER> WHEN READY"; E$
170  CLS: PRINT C$ " BOOKS" : PRINT
180  OPEN "I", #-1, "BOOKS"
190  IF EOF (-1) THEN 230
200  INPUT #-1, B$, A$, S$
210  IF S$ = C$ THEN PRINT B$, A$
220  GOTO 190
230  CLOSE #-1
```

Run the input part of the program by typing RUN 130 (ENTER). If you choose "fiction," this happens:

```
WHICH SUBJECT? FICTION
REWIND THE TAPE - PRESS PLAY
PRESS <ENTER> WHEN READY

FICTION BOOKS:

CAT'S CRADLE        KURT VONNEGUT
STEPPENWOLF         HERMANN HESSE
```

DO-IT-YOURSELF PROGRAM 25-1

Assume you have these checks:

NO.	DATE	PAYABLE TO	ACCOUNT	AMOUNT
101	5/13	Safeway	food	$52.60
102	5/13	Amoco	car	32.70
103	5/14	Joe's Cafe	food	10.32
104	5/17	American Airlines	vacation	97.50
105	5/19	Holiday Inn	vacation	72.30

Write a program that outputs all the checks to tape. Then have it input them from tape so that you can type one account—such as food—and the computer will tell you the total amount you've spent on food.

See "Sample Programs" in the Appendix for examples of how to store data on tape.

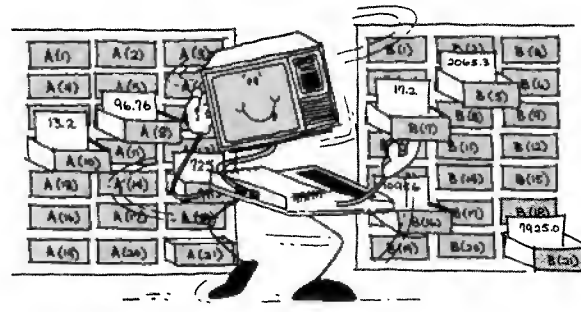
Learned in Chapter 25

BASIC WORDS	BASIC CONCEPT
OPEN	data files
CLOSE	
PRINT #-1	
INPUT #-1	
EOF	

Notes

MANAGING NUMBERS

Have you tried to write programs to handle much information? If so, you'll be glad to know Color BASIC has an easy-to-manage way to keep track of information.



Assume, for example, you want to write a program that lets you manage this information:

ELECTION RESULTS

District	Votes for Candidate A
1	143
2	215
3	125
4	331
5	442
6	324
7	213
8	115
9	318
10	314
11	223
12	152
13	314
14	92

Up to now, you've used variables to store information in memory. For example, to store the votes of the first three districts, type:

```
A = 143 (ENTER)
B = 215 (ENTER)
C = 125 (ENTER)
```

But there's a better kind of variable you can use. Type:

```
A(1) = 143 (ENTER)
A(2) = 215 (ENTER)
A(3) = 125 (ENTER)
```

Each of the above variables has a "subscript"—(1), (2), and (3). Other than how they use the subscript, these variables work the same as any other variables. To see for yourself, type both of these lines:

```
PRINT A; B; C (ENTER)
PRINT A(1); A(2); A(3) (ENTER)
```

Now take a quick look and compare the two programs below. Both work the same: Program 1 uses "simple variables"; Program 2 uses "subscripted variables."

PROGRAM 1

```
10  DATA 143, 215, 125, 331, 442
20  DATA 324, 213, 115, 318, 314
30  DATA 223, 152, 314, 92
40  READ A, B, C, D, E
50  READ F, G, H, I, J
60  READ K, L, M, N
70  INPUT "DISTRICT NO. (1-14)"; Z
75  IF Z>14 THEN 70
80  IF Z=1 THEN PRINT A "VOTES"
90  IF Z=2 THEN PRINT B "VOTES"
100  IF Z=3 THEN PRINT C "VOTES"
110  IF Z=4 THEN PRINT D "VOTES"
120  IF Z=5 THEN PRINT E "VOTES"
130  IF Z=6 THEN PRINT F "VOTES"
140  IF Z=7 THEN PRINT G "VOTES"
150  IF Z=8 THEN PRINT H "VOTES"
160  IF Z=9 THEN PRINT I "VOTES"
170  IF Z=10 THEN PRINT J "VOTES"
180  IF Z=11 THEN PRINT K "VOTES"
190  IF Z=12 THEN PRINT L "VOTES"
200  IF Z=13 THEN PRINT M "VOTES"
210  IF Z=14 THEN PRINT N "VOTES"
220  GOTO 70
```

PROGRAM 2

```
10  DATA 143, 215, 125, 331, 442
20  DATA 324, 213, 115, 318, 314
30  DATA 223, 152, 314, 92
40  DIM A(14)
50  FOR X = 1 TO 14
60  READ A(X)
70  NEXT X
80  INPUT "DISTRICT NO(1-14)"; Z
85  IF Z > 14 THEN 80
90  PRINT A(Z) "VOTES"
100  GOTO 80
```

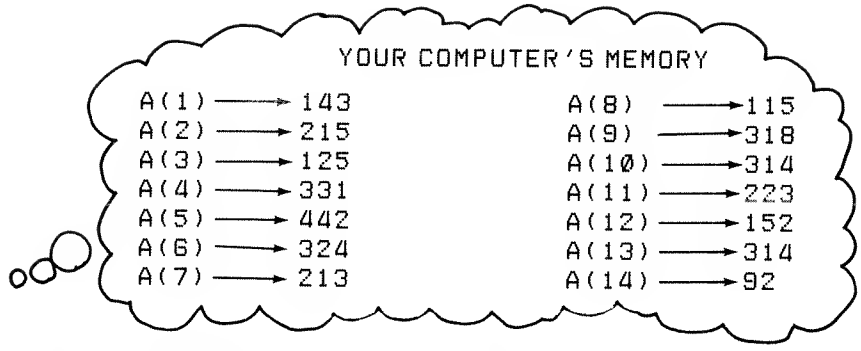
Program 1 is cumbersome to write. Program 2 is short and simple to write.

Enter and run Program 2. Here's how it works:

Line 40 reserves space for a list of information—called an "array" named A—with 14 subscripted items.

Lines 50 and 70 set up a loop to count from 1 to 14. Line 60 reads all 14 votes into Array A:

Actually, this leaves room for 15 subscripted items when you count 0 as a subscript.



Line 80 asks you to input a subscript, and Line 90 prints the item you requested.

The name of the array is A. The X or Z in parentheses refers to the subscript of one of the items.

Now that you've stored information in an array, it's easy to manage it. For instance, you can add these lines, which let you change the information:

```
92  INPUT "DO YOU WANT TO ADD TO THIS"; R$
94  IF R$ = "NO" THEN 80
96  INPUT "HOW MANY MORE VOTES"; X
97  A(Z) = A(Z) + X
98  PRINT "TOTAL VOTES FOR DISTRICT" Z "IS
    NOW" A(Z)
```

You don't need to study these programs if you're anxious to move on. We're just showing some benefits of using subscripted variables.

Or you can add these lines to display the information:

```
72  INPUT "DO YOU WANT TO SEE ALL THE TOTALS";
    S$
74  IF S$ = "YES" THEN GOSUB 110
100  GOTO 72
110  PRINT "DISTRICT", "VOTES"
120  FOR X = 1 TO 14
130  PRINT X, A(X)
140  NEXT X
150  RETURN
```

A Second Array

Assume you also want to keep track of a second candidate's votes—Candidate B:

ELECTION RESULTS

District	Votes for Candidate A	Votes for Candidate B
1	143	678
2	215	514
3	125	430
4	331	475
5	442	302
6	324	520
7	213	613
8	115	694
9	318	420
10	314	518
11	223	370
12	152	412
13	314	460
14	92	502

To do this, add another array to the program. Call it Array B. The following program records the votes for Candidate A (Array A) and Candidate B (Array B):

```
10  DATA 143, 215, 125, 331, 442
20  DATA 324, 213, 114, 318, 314
30  DATA 223, 152, 314, 92
40  DATA 678, 514, 430, 475, 302
50  DATA 520, 613, 694, 420, 518
60  DATA 370, 412, 460, 502
70  DIM A(14), B(14)
80  FOR X = 1 TO 14
90  READ A(X)
100  NEXT X
110  FOR X = 1 TO 14
120  READ B(X)
130  NEXT X
140  INPUT "DISTRICT NO.";Z
145  IF Z > 14 THEN 140
150  INPUT "CANDIDATE A OR B";R$
160  IF R$ = "A" THEN PRINT A(Z)
170  IF R$ = "B" THEN PRINT B(Z)
180  GOTO 140
```

DO-IT-YOURSELF PROGRAM 26-1

Write an inventory program that keeps track of 12 items (numbered 1-12) and the quantity you have of each item.

Deal the Cards

To keep track of 52 "cards," you need to use an array. Erase your program and type and run this one:

```
40  FOR X = 1 TO 52
50  C = RND(52)
90  PRINT C;
100  NEXT X
```

The computer deals 52 random "cards," but if you look closely, you see that some of the cards are the same.

To make sure the computer deals each card only once, you can build another array—Array T—that keeps track of each card dealt. Add these lines:

```
5  DIM T(52)
10  FOR X = 1 TO 52
20  T(X) = X
30  NEXT X
```

You don't need a DIM line if none of your array items use a label higher than 10. However, it's still a good idea to put this line in your program to reserve just the right amount of memory.

The above lines build Array T and put all 52 cards in it: T(1) = 1, T(2) = 2, T(3) = 3 . . . T(52) = 52.

Then add some lines that "erase" each card in Array T after it's dealt. Type:

```
60  IF T(C) = 0 THEN 50
80  T(C) = 0
```

Now the computer can't deal the same random card twice. For example, assume the computer first deals a two. Line 80 changes T(2)'s value from 2 to 0.

Then assume the computer deals another two. Since T(2) now equals 0, Line 60 goes back to Line 50 to deal another card.

Run the program. Note how the computer slows down at the end of the deck. It must try many different cards before it finds one that it hasn't dealt yet.

To play a card game, you need to keep track of which cards have been dealt. You can do this by building another array—Array D. Add these lines, which store all the cards, in the order they are dealt, in Array D:

```
7  DIM D(52)
70  D(X) = T(C)
90  PRINT D(X);
```

DO-IT-YOURSELF PROGRAM 26-2

Add lines to the program so that it displays only your "hand"—the first 5 cards dealt.

Learned in Chapter 26

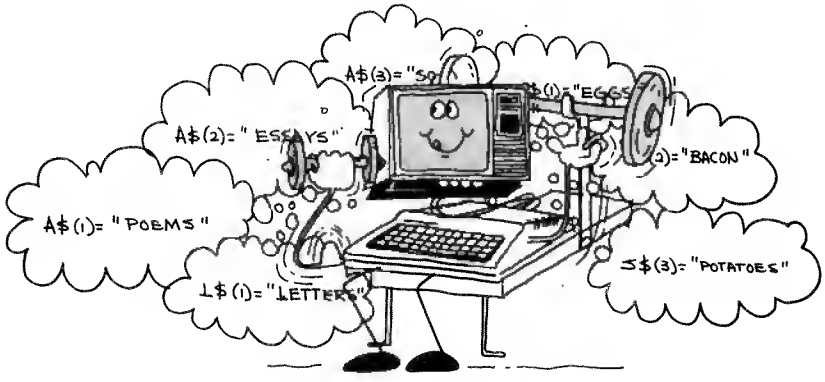
BASIC WORD	BASIC CONCEPT
DIM	arrays

Notes

CHAPTER 27

MANAGING WORDS

In the last chapter, you used arrays to manage numbers. Here, you'll use arrays to manage words by editing, updating, and printing an entire essay.



Start with a simple list of words: a shopping list:

1. EGGS
2. BACON
3. POTATOES
4. SALT
5. SUGAR
6. LETTUCE
7. TOMATOES
8. BREAD
9. MILK
10. CHEESE
11. FISH
12. JUICE

The dollar sign's the only difference between these subscripted variables and the ones in the last chapter.

Assign each word to a subscripted variable—this time use a subscripted *string* variable. For example, for the first three items, type:

```
S$(1) = "EGGS" ENTER
S$(2) = "BACON" ENTER
S$(3) = "POTATOES" ENTER
```

To see how the items are stored, type:

```
PRINT S$(1), S$(2), S$(3) ENTER
```

Now build a program that reads these words into an array named S$ and then displays them:

```
5  DIM S$(12)
10  DATA EGGS, BACON, POTATOES, SALT
20  DATA SUGAR, LETTUCE, TOMATOES, BREAD
30  DATA MILK, CHEESE, FISH, JUICE
40  FOR X = 1 TO 12
50  READ S$(X)
60  NEXT X
70  PRINT "SHOPPING LIST:"
80  FOR X = 1 TO 12
90  PRINT X; S$(X)
100  NEXT X
```

DO-IT-YOURSELF PROGRAM 27-1

Add some lines to the above program so that you can change any item on this list.

DO-IT YOURSELF PROGRAM 27-2

Here is a program that uses an array to write song lyrics.

```
5  DIM A$(4)
10  PRINT "TYPE 4 LINES"
20  FOR X = 1 TO 4
30  INPUT A$(X)
40  NEXT X
50  CLS
60  PRINT "THIS IS YOUR SONG:"
70  PRINT
80  FOR X = 1 TO 4
90  PRINT X; " "; A$(X)
100  NEXT X
```

Add some lines so that you can revise any line.

Want to compose music? Look up "Music Composer" in the "Sample Programs" appendix.

Haven't heard of word processing? It's a kind of program that lets you type and store information, make changes to it, and print it out on demand.

Need a refresher on some of this? CLEAR is in Chapter 8 and INKEY$ is in Chapter 11.

Writing an Essay (. . . A Novel, Term Paper . . .)

Now that you've learned how to use string arrays, it will be easy to write a program that stores and edits what you type. Type this program:

```
1  CLEAR 1000
5  DIM A$(50)
10  PRINT "TYPE A PARAGRAPH"
20  PRINT "PRESS </> WHEN FINISHED"
30  X = 1
40  A$ = INKEY$
50  IF A$ = "" THEN 40
60  PRINT A$;
70  IF A$ = "/" THEN 110
80  A$(X) = A$(X) + A$
90  IF A$ = "," THEN X = X + 1
100  GOTO 40
110  CLS
120  PRINT "YOUR PARAGRAPH:"
130  PRINT
140  FOR Y = 1 TO X
150  PRINT A$(Y);
160  NEXT Y
```

Run the program. To see how each sentence is stored, type these lines:

```
PRINT A$(1) ENTER
PRINT A$(2) ENTER
PRINT A$(3) ENTER
```

Here's how the program works:

Line 1 clears plenty of string space.

Line 5 saves room for an array named A$ that may have up to 50 sentences.

Line 30 makes X equal to 1. X will be used to label all the sentences.

Line 40 checks to see which key you are pressing. If it is nothing (" "), Line 50 sends the computer back to Line 40.

Line 60 prints the key you pressed.

Line 70 sends the computer to the lines that print your paragraph when you press the "/" key.

Line 80 builds a string and labels it with number X. X is equal to 1 until you press a period (.). Then Line 80 makes X equal to X + 1.

For example, if the first letter you press is "R,"

A$(1) EQUALS "R".

If the second letter you press is "O",

A$(1) EQUALS A$(1) - WHICH IS "R" + "O"
OR
"RO".

Assume that when A$(1) equals ROSES ARE RED, you press a period. A$(1) then equals the entire sentence: ROSES ARE RED. The next letter you press is in A$(2).

Lines 140–160 print your paragraph.

DO-IT-YOURSELF CHALLENGER PROGRAM 27-3

Here's a tough one (but it can be done!) for those intrigued with word processing. Change the above program so that you can:

1. Print any sentence
2. Revise any sentence

You may need to review the challenger program in Chapter 12. Our answer's in the back.

Using the Printer

If you have a printer, connect it now by plugging it into the jack marked SERIAL I/O. Turn on the printer and insert paper. The manual that comes with the printer shows how.

Ready? Type this short program:

```
10  INPUT A$
20  PRINT # - 2, A$
```

Now type:

```
LLIST (ENTER)
```

If your program doesn't list on the printer, be sure the printer is on, "on-line," and connected to your keyboard. Then type `LLIST` (ENTER) again.

Run the program and watch the printer work. `PRINT # - 2`, tells the computer to print, not on the screen, but on device # - 2, which is the printer. Be sure to type a comma after the -2, or you get a syntax error.

Having trouble getting into this mode? Read the end of Chapter 1.

Press the (SHIFT) and (0) (zero) keys simultaneously and release them so that the letters you type appear in reversed colors on your screen (green with a black background). You are now in an upper- lowercase mode. The reversed colored letters are actually lowercase (noncapitalized) letters.

To type a capital letter, use the (SHIFT) key as you do with a typewriter. It appears in regular colors.

Run the program, using the (SHIFT) key so that the word RUN is capitalized. Input a sentence with both upper- and lowercase letters. Type:

`MY PRINTER PRINTS LOWERCASE LETTERS (ENTER)`

All the letters in RUN should appear in regular (not reversed) colors.

If you have a Deluxe Color Computer, you can get true lower-case letters (rather than reversed letters) to appear on your screen. See ***Introducing Your Deluxe Color Computer****.*

DO-IT-YOURSELF PROGRAM 27-4

Look at the "Writing an Essay" program earlier in this chapter. Change Lines 140-160 so that the paragraph prints on the printer rather than the screen.

Learned in Chapter 27

BASIC WORDS	BASIC CONCEPT
LLIST	string arrays
PRINT # - 2	

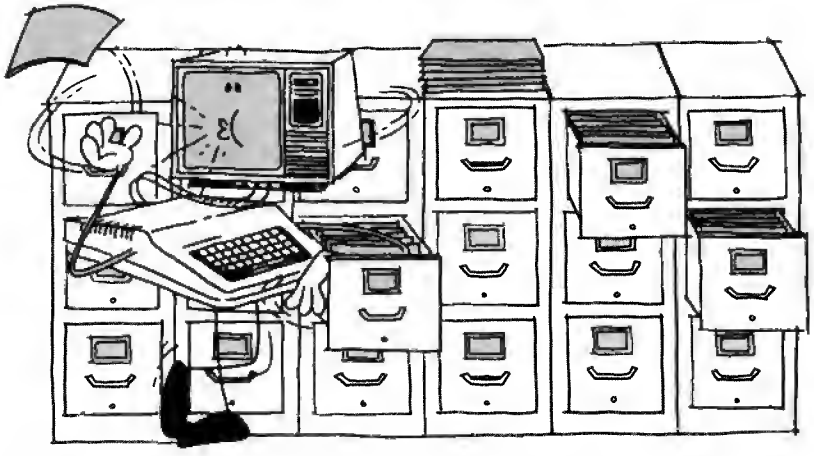
Notes

CHAPTER 28

SORTING

Any file clerk knows it's easier to find information that's sorted alphabetically. Type this program and run it, until you're convinced the computer can alphabetize:

```
10  INPUT "TYPE TWO WORDS"; A$, B$
20  IF A$ < B$ THEN PRINT A$ " COMES BEFORE " B$
30  IF A$ > B$ THEN PRINT A$ " COMES AFTER " B$
40  IF A$ = B$ THEN PRINT "BOTH WORDS ARE THE
    SAME"
50  GOTO 10
```



With strings, the greater than (>), less than (<), and equal (=) signs have a new meaning. They tell which of two strings comes before the other in alphabetical sequence:

<	precedes alphabetically
<=	precedes or is the same alphabetically
>	follows alphabetically
>=	follows or is the same alphabetically
=	is the same

Since the computer can alphabetize, it's easy to write a sorting program. Type and run this program, which sorts 5 words:

```
10  DIM A$(5)
20  FOR I = 1 TO 5
30  INPUT "TYPE A WORD"; A$(I)
40  NEXT I
50  X = 0
60  X = X + 1
70  IF X > 5 THEN GOTO 70
80  IF A$(X)="ZZ" THEN 60
90  FOR Y = 1 TO 5
100   IF A$(Y) < A$(X) THEN X = Y
110   NEXT Y
120   PRINT A$(X)
130   A$(X)="ZZ"
140   GOTO 50
```

You can easily make the computer alphabetize more words by changing the 5 to say, 100, in Lines 10, 20, 70, and 90.

To see how the program works, delete Line 120 and add the following lines. (These lines only show what the program does—they have nothing to do with sorting.)

```
120
5  CLS
45 CLS
85  V = V + 1
105  PRINT @ 15+32*(V-1), A$(X)
135  GOSUB 500
500  FOR I = 1 TO 5
510 PRINT @ 0+32*(I-1),A$(I);"      ";
520  NEXT I
530  RETURN
```

Run the program. Too fast? Type this line. It slows down the program so you can see what's happening:

```
107  FOR T = 1 TO 600: NEXT T
```

Now run the program again. Input these words and watch carefully:

```
MICHAEL
TRAVIS
DYLAN
ALEXIA
SUSAN
```

Look at Column 2. See how the first name changes from Michael to Dylan to Alexia. Next, notice what happens to Alexia in the first column. Alexia becomes ZZ.

This illustrates how the program sorts the first and second words:

FIRST WORD

```
                MICHAEL          MICHAEL
TRAVIS                           TRAVIS
DYLAN           DYLAN
ALEXIA          ALEXIA           ALEXIA
SUSAN           SUSAN            SUSAN

MICHAEL         MICHAEL          MICHAEL ALEXIA
TRAVIS          TRAVIS           TRAVIS
DYLAN           DYLAN            DYLAN
                ALEXIA           ZZ
SUSAN                            SUSAN
```

SECOND WORD

```
        ALEXIA  MICHAEL ALEXIA   MICHAEL ALEXIA
TRAVIS                           TRAVIS
DYLAN           DYLAN
ZZ              ZZ               ZZ
SUSAN           SUSAN            SUSAN

MICHAEL ALEXIA  MICHAEL ALEXIA   MICHAEL ALEXIA
TRAVIS          TRAVIS           TRAVIS  DYLAN
DYLAN           DYLAN            ZZ
                ZZ               ZZ
SUSAN                            SUSAN
```

Here's how the program works:

Lines 50 and 60 set X's value. At the start, X is 1.

Then Lines 90–110 compare A$(X)—Michael—with every other name in Array A$ until a word is reached that precedes Michael—Dylan.

Line 100 then makes A$(X) equal to Dylan's place in the array: A$(3). When Dylan is compared with the fourth word—Alexia—A$(X) becomes A$(4).

When all the words have been compared with one another, Line 120 displays the first sorted word: Alexia. Line 130 changes Alexia's position—A$(4)—to ZZ.

At this point, Lines 50 and 60 make X equal 1 again. A$(X)—Michael—is compared with other names in the array to find the second sorted word.

When Michael's place in the array becomes ZZ, Line 60 sets X to 2. Then, A$(X)—which is now Travis—is compared with all the names in the array to find the next sorted word.

When the array's values are all changed to ZZ, Line 70 ends the program.

DO-IT-YOURSELF PROGRAM 28-1

Using this sort routine, change the program from the last chapter so that it alphabetizes your books by title, author, or subject.

This chapter shows a simple way to sort. If you need to sort many items, you may want to research faster sorting methods (such as the bubble sort).

Learned in Chapter 28

BASIC SYMBOLS

>
<
=

Notes

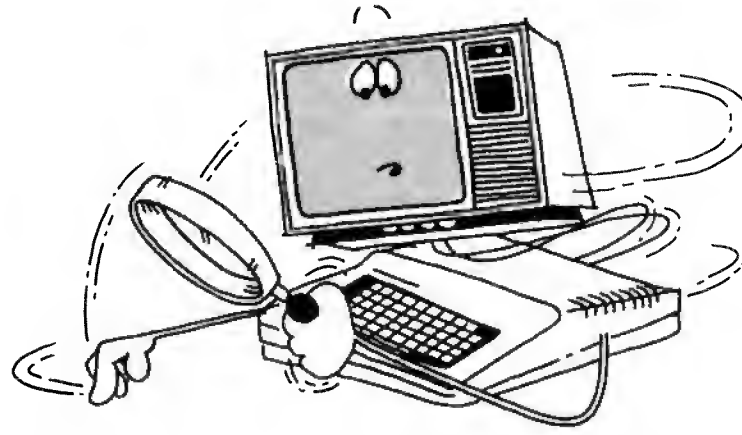
_____

_____

_____

ANALYZING

If you have more than 4K RAM, you have an easy way to analyze information. By giving each item more than one subscript, you can see it through different dimensions.



Take the voting program from Chapter 19. Here's the information. (We're using only the first three districts to make the program simple.)

We're only using three districts to keep it simple.

We're calling them Candidates 1 and 2 this time rather than Candidates A and B.

ELECTION POLL

District	Votes for Candidate 1	Votes for Candidate 2
1	143	678
2	215	514
3	125	430

In Chapter 19, you stored the above "items" (groups of votes) in *two one-dimensional arrays*: Arrays A and B. In this chapter, you'll store them in *one* easy-to-manage *two-dimensional array*: Array V.

The following program puts the items in Array V.

```
5   DIM V(3,2)
10   DATA 143, 678, 215, 514, 125, 430
20   FOR D = 1 TO 3
30   FOR C = 1 TO 2
40   READ V(D,C)
50   NEXT C
60   NEXT D

70   INPUT "DISTRICT NO. (1-3)"; D
80   IF D < 1 OR D > 3 THEN 70
90   INPUT "CANDIDATE NO. (1-2)"; C
100   IF C < 0 OR C > 2 THEN 90
110   PRINT V(D,C)
120   GOTO 70
```

Type and run the program. Notice that each item is labeled by two subscripts.

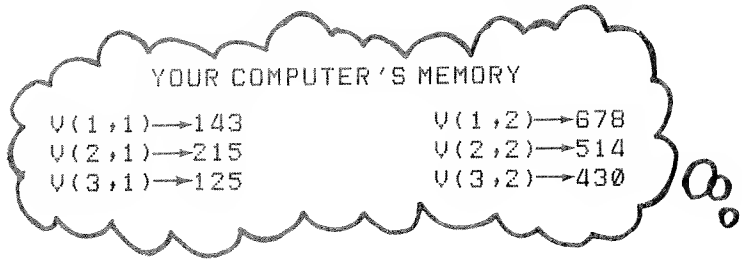
Here's how the program works:

Line 5 reserves space in memory for Array V. Each item in Array V can have two subscripts: the first, no higher than 3; the second, no higher than 2.

Lines 20–60 read all the votes into Array V, giving them each two subscripts:

The first subscript is the district (Districts 1-3).

The second subscript is the candidate (Candidates 1-2).



For example, 678 is labeled V(1,2). This means 678 is from District 1 and is for Candidate 2.

With all the votes in a two-dimensional array, it's simple to analyze them—in two dimensions. By adding these lines, for example, you can print all the votes in two ways: by district and by candidate.

(Delete Lines 70-120 first)

```
70   INPUT "TYPE < 1 > FOR DISTRICT OR
     < 2 > FOR CANDIDATE"; R
80   IF R < 1 OR R > 2 THEN 70
100   ON R GOSUB 1000, 2000
110   GOTO 70

1000   INPUT "DISTRICT NO(1-3)"; D
1010   IF D < 1 OR D > 3 THEN 1000
1015   CLS
1020   PRINT @ 132, "VOTES FROM DISTRICT" D
1030   PRINT
1040   FOR C = 1 TO 2
1050   PRINT "CANDIDATE" C,
1060   PRINT V(D,C)
1070   NEXT C
1080   RETURN

2000   INPUT "CANDIDATE NO(1-2)"; C
2010   IF C < 1 OR C > 2 THEN 2000
2015   CLS
2020   PRINT @ 132, "VOTES FOR CANDIDATE" C
2030   PRINT
2040   FOR D = 1 TO 3
2050   PRINT "DISTRICT" D,
2060   PRINT V(D,C)
2070   NEXT D
2080   RETURN
```

Remember how to delete lines? 70 (ENTER) Deletes Line 70.

If you are truly an analytical type, you're going to love the rest of this chapter. If you're definitely NOT that type, skip it!

The Third Dimension

You can continue with as many dimensions as you want. You're limited only by how much information you can fit into the computer's memory.

Add a third dimension to Array V: interest groups. Here's the information:

VOTES FROM INTEREST GROUP 1

	Candidate 1	Candidate 2
District 1	143	678
District 2	215	514
District 3	125	430

VOTES FROM INTEREST GROUP 2

	Candidate 1	Candidate 2
District 1	525	54
District 2	318	157
District 3	254	200

VOTES FROM INTEREST GROUP 3

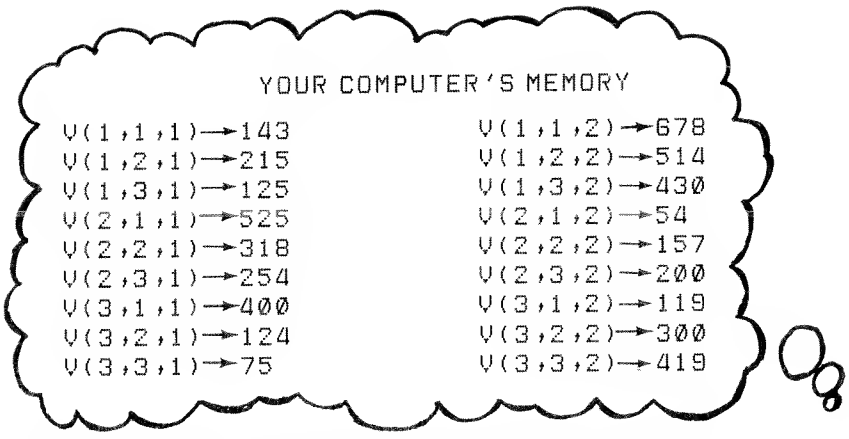
	Candidate 1	Candidate 2
District 1	400	119
District 2	124	300
District 3	75	419

To get all this into your computer's memory, erase your program and type:

```
5  DIM V(3,3,2)
10  DATA 143, 678, 215, 514, 125, 430
20  DATA 525, 54, 318, 157, 254, 200
30  DATA 400, 119, 124, 300, 75, 419

40  FOR G = 1 TO 3
50  FOR D = 1 TO 3
60  FOR C = 1 TO 2
70  READ V(G,D,C)
80  NEXT C
90  NEXT D
110  INPUT "INTEREST GROUP NO (1-3)"; G
120  IF G < 1 OR G > 3 THEN 110
130  INPUT "DISTRICT NO. (1-3)"; D
140  IF D < 1 OR D > 3 THEN 130
150  INPUT "CANDIDTE NO. (1-2)"; C
160  IF C < 1 OR C > 2 THEN 150
170  PRINT V(G,D,C)
180  GOTO 110
```

Run the program and test the subscripts. Lines 40-100 read all the votes into Array V, giving them each three subscripts:

- The first subscript is the interest group (Interest Groups 1-3).
- The second subscript is the district (Districts 1-3).
- The third subscript is the candidate (Candidates 1-2).



For example, 678 is now labeled V(1,1,2). This means 678 is from Interest Group 1, is from District 1, and is for Candidate 2.

To take advantage of all three dimensions, delete Lines 110-180 and type:

```
110  PRINT: PRINT "TYPE <1> FOR GROUP"
120  PRINT "<2> FOR DISTRICT OR <3> FOR
     CANDIDATE"
130  P = 224 : INPUT R
140  ON R GOSUB 1000,2000,3000
150  GOTO 110

1000  INPUT "GROUP(1-3)"; G
1010  IF G<1 OR G>3 THEN 1000
1020  CLS
1030  PRINT @ 102, "VOTES FROM GROUP" G
1040  PRINT @ 168, "CAND. 1"
1050  PRINT @ 176, "CAND. 2"
1060  FOR D = 1 TO 3
1070  PRINT @ P, "DIST." D
1080  FOR C = 1 TO 2
1100  PRINT @ P + 8*C, V(G,D,C);
1110  NEXT C
1120  P = P + 32
1130  NEXT D
1140  RETURN

2000  INPUT "DISTRICT(1-3)"; D
2010  IF D<1 OR D>3 THEN 2000
2020  CLS
2030  PRINT @ 102, "VOTES FROM DIST." D
2040  PRINT @ 168, "CAND. 1"
2050  PRINT @ 176, "CAND. 2"
2060  FOR G = 1 TO 3
2070  PRINT @ P, "GROUP" G
2080  FOR C = 1 TO 2
2100  PRINT @ P + 8*C,V(G,D,C);
2110  NEXT C
2120  P = P + 32
2130  NEXT G
2140  RETURN

3000  INPUT "CANDIDATE(1-2)"; C
3010  IF C<1 OR C>2 THEN 3000
```

```
3020  CLS
3030  PRINT @ 102, "VOTES FOR CAND." C
3040  PRINT @ 168, "DIST. 1"
3050  PRINT @ 176, "DIST. 2"
3060  PRINT @ 184, "DIST. 3"
3070  FOR G = 1 TO 3
3080  PRINT @ P, "GROUP" G
3090  FOR D = 1 TO 3
3100  PRINT @ P + 8*D, V(G,D,C);
3110  NEXT D
3120  P = P + 32
3130  NEXT G
3140  RETURN
```

Run the program. You can now get three perspectives on the information.

DO-IT-YOURSELF PROGRAM 29-1

Write a program to deal the cards using a two-dimensional array. Make the first dimension the card's suit (1-4) and the second dimension the card's value (1-13).

Learned in Chapter 29

BASIC CONCEPT

Multidimensional arrays

Notes

SECTION IV

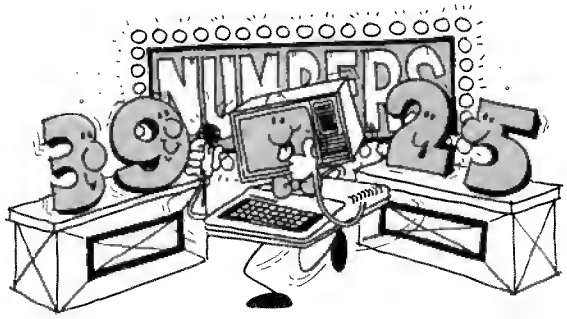
BACK TO BASICS

This section sends you back to school. You'll learn some new Extended Color BASIC words that will help you refine and polish your programs.

CHAPTER 30

THE NUMBERS GAME

Your "extended" Color Computer includes several advanced mathematical functions. This chapter gives a rundown of each function and shows the ways to use it.



Before continuing, however, you need to know about a couple of functions and definitions discussed below.

Exponentiation

(↑)

Quick! What's 1.5 squared? How about 77 cubed? If you don't know, ask the computer. Anytime you want to raise a number to the *n*th power, follow this format:

number (↑) *power*

number is the base (the number you wish to raise to the *n*th power). It may be any numeric expression.

up-arrow is generated by pressing (↑).

power is the exponent to which the base is raised. It may be any numeric expression.

Note: Exponentiation has precedence over other operators. For example, if the computer calculates −2 up-arrow 2, the result is a negative number. To raise −2 to the 2d power "correctly" (resulting in positive number), enclose −2 in parentheses.

Start with 77 cubed. After looking at the syntax block, can you give the command? Your answer should be 456533.002.

If your screen looks like this, you're off to a good start:

```
PRINT 77 ↑ 3
456533.002
OK
```

Try raising 10 to the 10th power. The screen displays:

```
1.00000001E+10
```

Don't worry about the ".002." This is called a "round-off error" and is necessary because the computer isn't the "perfect" calculator. But then, no machine is.

Since 10,000,000,000 has more than 9 significant digits, the computer went into the E notation explained in Chapter 13.

How about 100 to the 100th power? Does the screen display an ?OV ERROR (overflow)? This means that the answer is too large for the computer to handle. Anything outside the range -10^{38} to $+10^{38}$ causes an overflow error.

> DO-IT-YOURSELF PROGRAM 30-1
>
> Write a short program that displays the square of each whole number from 1 to 10.

SQR

SQR enables you to find the square root of a number. Here is its syntax:

> **SQR** (*number*)
>
> *number* is zero or any positive number.

For example, if you want the square root of 100, type:

```
PRINT SQR(100) ENTER
```

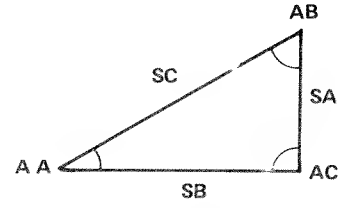
and you'll find out (if you didn't already know) that the answer is 10.

> DO-IT-YOURSELF PROGRAM 30-2
>
> Write another short program to display the square root of every tenth number from 100 to 0.

TRIG Functions

Trigonometry is the investigation of the relationship of a triangle's sides to its angles.

Look at this triangle. You'll be using it throughout the discussion of trigonometric functions.



Trigonometry has many practical applications. For instance, imagine that your triangle is actually the roof of a house you're building. Trigonometric functions can help you determine either the length of the rafters or the slope of the roof (the "pitch"). So if math turns you off but building things turns you on, this section might be just what you're looking for.

Notice that we've labeled angles with the prefix A and sides with the prefix S. Angle A, for example, is AA; the side opposite it is SA.

Using the triangle, we can define the common trig functions in the following manner:

Sine of AA = SIN (AA) = SA/SC

Cosine of AA = COS (AA) = SB/SC

Tangent of AA = TAN (AA) = SA/SB

Degrees *v* Radians

To define an angle, you may use either of two units of measurement. The more common unit is the degree; the "more technical" unit is the radian.

Your computer assumes all angles are measured in radians. Since radians may be somewhat alien to you, you can convert them to degrees (and vice versa) this way:

Degrees to Radians: Degrees / 57.29577951
Radians to Degrees: Radians * 57.29577951

This chapter's sample programs include a "converter" that takes the degrees you input and automatically converts them into radians (and vice versa for some purposes).

SIN

That's sine—pronounced like "sign."

Its syntax is:

> **SIN (*angle*)**
>
> *angle* is angle's size in radians.

Given the length of one side and the sizes of two angles, you can use SIN to determine the lengths of the other sides.

Enter and run the following program, inputting any values.

```
5 CLS
10 INPUT "WHAT IS ANGLE A (AA)"; AA:
   IF AA<=0 OR AA>=180 THEN 100
20 INPUT "WHAT IS ANGLE B (AB)"; AB:
   IF AA <=0 OR AB >=180 THEN 100
30 INPUT "WHAT IS SIDE C (SC)"; SC:
   IF SC <=0 THEN 100
40 AC = 180-(AA+AB)  'VALUE OF ANGLE AC
50 IF (AA+AB+AC) < > 180 THEN 100
   'TRIANGLE=180 DEGREES
60 AA=AA/57.29577951: AB=AB/57.29577951:
   AC=AC/57.29577951
   ' CONVERT DEGREES TO RADIANS
70 SA=((SIN(AA))/(SIN(AC))) * SC: IF SA<0
   THEN 100
80 SB=((SIN(AB))/(SIN(AC))) * SC: IF SB<0
   THEN 100
90 PRINT "SIDE A (SA) IS" SA "LONG":
   PRINT "SIDE B(SB) IS"
   SB "LONG": GOTO 10
100 PRINT "SORRY, NOT A TRIANGLE.
    TRY AGAIN": GOTO 10
```

In the Sample Program section is a program called Drawing Triangles. That program draws triangles based upon sides and angles that you specify.

When the computer asks you for AB and AC, input degree-measures of the angles. If you enter a negative number or a number that is greater than or equal to 180, the computer goes to Line 100. It then prints the message and again asks for the sizes. If you enter a negative number for SC, it does the same thing.

Since you don't know the size of AC, the computer automatically computes this in Line 40. If the sum of the three angles is not equal to 180 degrees, the computer takes appropriate action in Line 50. Line 60 converts degrees to radians so the computer can do the sine calculations.

Sine Waves

You may have seen sine waves before. They're used to indicate AC power and other electrical conditions. Run the following program to see a "horizontal scrolling" sine wave (and check the Sample Program section for a more conventional sine wave).

```
5 CLS
10 FOR A = 180 TO -179 STEP-10
20 RD = A / 57.29577951   'RADIANS
30 CL = SIN(RD) * 14 + 16.5
   'CL = COLUMN POSITION
40 PRINT TAB(CL);"S"   'PLOT SINE OF RD
50 NEXT A
60 GOTO 60
```

COS

The cosine function is related to the sine function and has the following syntax:

COS (*angle*)

angle is angle's size in radians.

Given the lengths of two sides and the size of one angle, you can use cosine to determine the length of a triangle's third side, as shown here:

```
5 CLS
10 INPUT "WHAT IS ANGLE C (AC)"; AC:
   IF AC<=0 OR AC>=180 THEN 100
20 AC=AC / 57.29577951
   'CONVERT DEGREES TO RADIANS
30 INPUT "WHAT IS SIDE A (SA)"; SA:
   IF SA<=0 THEN 100
40 INPUT "WHAT IS SIDE B (SB)"; SB:
   IF SB=<0 THEN 100
50 SC = ((SA ↑ 2)+(SB ↑ 2))-(2*(SA*SB*
   COS(AC))): IF SC<0  THEN 100
60 PRINT "SIDE C (SC) IS" SQR(SC) "LONG":
   GOTO 10
100 PRINT "SORRY, NOT A TRIANGLE,
    TRY AGAIN": GOTO 10
```

Notice that the program works almost the same as the SIN program except for the use of exponentiation (up-arrow) in Line 50 and SQR in Line 60.

DO-IT-YOURSELF PROGRAM 30-3

Cosine can make waves of its own. Rewrite the "Sine Wave" program so that it plots COS(RD) instead of SIN(RD). Use C (for cosine) to display the wave made by COS. What is the difference between the two?

TAN

The third trigonometric function, TAN, lets you calculate the tangent of an angle. Here is its syntax:

TAN (*angle*)

angle is angle's size in radians.

You can use TAN to determine, among other things, one side of a triangle, given another side and one angle.

Enter and run this program:

```
5 CLS
10 INPUT "WHAT IS SIDE B (SB)"; SB:
   IF SB<=0 THEN 100
20 INPUT "WHAT IS ANGLE A (AA)"; AA:
   IF AA<=0 OR AA>=180  THEN 100
30 AA=AA/57.29577951   'CONVERT DEGREES
   TO RADIANS
40 SA=SB*(TAN(AA)): IF SA<=0 THEN 100
50 PRINT "SIDE A (SA) IS" SA "LONG":
   GOTO 10
100 PRINT "SORRY, NOT A TRIANGLE.
    TRY AGAIN": GOTO 10
```

The key to this program, of course, is Line 40, where the tangent of AA is multiplied by the length of SB to determine the length of SA.

ATN

ATN (arctangent) is the inverse of TAN and has the following syntax:

ATN (*angle*)

angle is angle's size in radians.

The following program uses ATN and TAN to calculate two unknown angles of a triangle when two sides and one angle are known.

```
10 CLS
20 INPUT "WHAT IS SIDE A (SA)"; SA:
   IF SA<=0 THEN 150
30 INPUT "WHAT IS SIDE C (SC)"; SC:
   IF SC<=0 THEN 150
40 INPUT "WHAT IS ANGLE B (AB)"; AB:
   IF AB<=0 OR AB>=180 THEN 150
50 X=(180-AB)   'AA+AC=180-AB
60 X=X/57.29577951   'CONVERT DEGREES
   TO RADIANS
70 Y=((SA-SC)/(SA+SC))*TAN(X/2)
```

```
80 Z=ATN(Y)
90 AA=(X/2)+(Z)
100 AC=(X/2)-(Z)
110 AA=AA*57.29577951   'CONVERT
    RADIANS TO DEGREES
120 AC=AC*57.29577951   'CONVERT RADIANS
    TO DEGREES
130 PRINT "ANGLE A (AA) IS" AA "DEGREES"
140 PRINT "ANGLE C (AC) IS" AC "DEGREES":
    GOTO 20
150 PRINT "SORRY, NOT A TRIANGLE.
    TRY AGAIN": GOTO 20
```

TAN ((AA-AC)/2) is equal to ((SA-SC)/(SA + SC)) * TAN ((AA + AC)/2). Also notice that it was necessary to convert the "computer's" radians to "your" degrees (Lines 110 and 120).

LOG

LOG returns the natural logarithm of a number. This is the inverse of EXP, so X = LOG(EXP(X)). Here is LOG's syntax:

LOG (*number*)

number is greater than zero.

The logarithm of a number is the power to which a given "base" must be raised to result in the number. "Logs" are useful in scientific and mathematical problems. In the LOG function, if you omit the base, the computer assumes you are specifying Base e (2.718281828).

To find the logarithm of a number to another base, B, use this formula:

log base B (x) = log e (x) / log e (B)

For example, LOG (32768)/LOG(2) returns the logarithm to Base 2 of 32768. (It returns the power to which 2 is raised to get 32768.)

Try these:

```
PRINT LOG (1) ENTER
PRINT LOG (100) ENTER
PRINT LOG (2.718281828) ENTER
```

DO-IT-YOURSELF PROGRAM 30-4

Compute the LOG of each of the following numbers:

a) 1003 b) 74.9865 c) 3.354285

DO-IT-YOURSELF PROGRAM 30-5

Compute the log to Base 10 of each of the following numbers:

a) 1 b) 10 c) 100
d) 500 e) 0.1 f) 1001

log e x
Hint: log 10 x = _____
log e 10

EXP

The EXP function returns the natural exponential of a number (*enumber*). EXP is the inverse of LOG; therefore, X = EXP(LOG(X)). Here is EXP's syntax:

EXP (*number*)

number is less than 87.3365.

Run this program to see EXP at work.

```
10 CLS
20 INPUT "ENTER X"; X
30 PRINT "EXP(X)="; EXP(X)
40 GOTO 20
```

FIX

It's impressive when your computer carries a number out to 9 significant digits, especially when 8 of those numbers are to the right of the decimal point.

However, sometimes you might not want all those numbers; you may want only the whole-number portion (the number to the left of the decimal point). FIX lets you get this whole number by simply chopping off all digits to the right of the decimal point. Here is FIX's syntax:

FIX (*number*)

For example, type:

```
PRINT FIX (2.2643951) ENTER
```

The computer displays:

```
2
OK
```

Here's a program that breaks a number into its whole and fractional portions.

```
10 CLS
20 INPUT "A NUMBER LIKE X.YZ"; X
30 W=FIX(X)
40 F=ABS(X)-ABS(W)
50 PRINT "WHOLE PART="; W
60 PRINT "FRACTIONAL PART="; F
70 GOTO 20
```

DEF FN

Extended Color BASIC has one numeric function, DEF FN, that is unlike any others we've talked about so far. DEF FN lets you create your own mathematical function. You can use your new function the same as any of the available functions (SIN, COS, TAN, and so on). Once you've used DEF FN to define a function, you may put it to work in

When you use this feature, don't forget to use the DEF FN statement before you try to execute the function it defines. Otherwise a ?UF ERROR (undefined function) occurs.

your program by attaching the prefix FN to the name you assign to the new function. Here is the syntax for DEF FN:

DEF FN *name (variable list)=formula*

name is the name you assign to the function you create.

variable list contains one "dummy variable" for each variable to be used by the function.

formula defines the operation in terms of the variables given in the variable list

Note: Variable names that appear in *formula* serve only to define the *formula*; they do not affect program variables that have the same name. You may have only one argument in a formula call; therefore, DEF FN must contain only one variable.

You may use DEF FN only in a program, not in the immediate mode.

For example, one math operation you've had to use several times in this chapter is degree-to-radian conversion. Wouldn't it be nice if the computer did that for you?

If you'll change the sample program we used for SIN, you'll see how to create a DEF FN that converts degrees to radians.

```
7 DEF FNR(X)=X/57.29577951
60 AA=FNR(AA): AB=FNR(AB): AC=FNR(AC)
```

You can see right away how much typing this saves, since you had to enter 57.29577951 only once. Whenever FNR is called into use, the computer automatically inserts whatever values you have used and performs the prescribed operation.

DO-IT-YOURSELF PROGRAM 30-6

Use DEF FN to:

1. Convert radians to degrees.
2. Create a math function that cubes numbers.

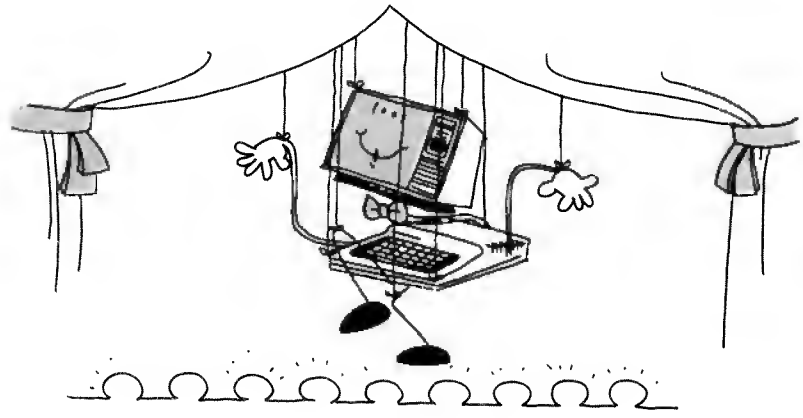
You'll find a quick reference table of many useful mathematical formulas (plane geometry, trig, and algebra) in the Appendix.

Learned in Chapter 30

BASIC WORDS	CONCEPTS
SQR	Computing a square root
SIN	Computing the sine; Determining two unknown sides of a triangle, given two angles and a side.
COS	Computing the cosine; Determining the unknown side of a triangle, given two sides and an angle
TAN	Computing the tangent; Determining the unknown side of a triangle, given one side and an angle
ATN	Computing the arctangent; Determining two unknown angles of a triangle, given two sides and the third angle
LOG	Computing the natural logarithm of a number
EXP	Computing the natural exponential of a number
FIX	Rounding a decimal number to a whole number
DEF FN	Defining a function

Notes

IT DON'T MEAN A THING IF IT AIN'T GOT THAT STRING



Earlier, we discussed string at great length. Now it's time for information about more of Extended Color BASIC's string functions.

STRING$

Zing goes STRING$ . . . and when you use it to create a string of characters, you can produce graphs, tables, and any other text display. The syntax of STRING$ is as follows:

STRING$ (*length,character*)

length is a number from 0 to 255.

character is either a string expression for a character or a numeric expression for an ASCII code. If you use a string constant, enclose it in quotes.

The number of characters displayed depends on the number you specify in *length*. Which characters are used depends on either the character or the ASCII code you specify. See the Appendix for a complete list of ASCII character codes.

For instance, jazz up your overworked "Lines" program by changing it as follows:

```
5 CLS
6 X$ = STRING$ (13,"*")
7 PRINT @ 96, X$; "LINES"; X$
9 FOR X = 1 TO 1000: NEXT X
10 PMODE 3,1
15 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
40 GOTO 40
```

Line 6 assigns X$ the value STRING$ (13,"*")—a string of 13 asterisks.

Line 7 tells the computer to print (starting at Print Screen Location 96) X$, then the word LINES, followed by X$ again. (See the Text Screen Worksheet in the Appendix.) Since X$ equals 13 asterisks (*), those characters are printed before and after LINES.

What? You want to spruce up the title even more! All right, add these two lines:

```
8 Y$ = STRING$(31,42): PRINT @ 384,Y$
```

This time, you tell the computer to display the character represented by ASCII Code 42. And, as you probably guessed, ASCII Code 42 represents an asterisk.

DO-IT-YOURSELF PROGRAM 31-1

Have you ever written lists to check off jobs that you or other people have to do?

Using STRING$, write a program that creates a check-off list.

I Think I See Some-String Ahead!
(INSTR)

If you want to search through one string for a second string, use INSTR.

INSTR's syntax is:

INSTR (*position,search-string,target*)

position specifies the position in the *search-string* at which the search is to begin (0 to 255). If you omit *position*, the computer automatically begins at the first character.

search-string is the string to be searched.

target is the string for which to search.

INSTR returns a 0 if any of the following is true:

The *position* is greater than the number of characters in the *search-string*.

The *search-string* is null.

It cannot find the *target*.

Watch the way INSTR works in the following program:

```
5 CLEAR 500
10 CLS
15 INPUT "SEARCH TEXT";S$
20 INPUT "TARGET TEXT";T$
25 C=0: P=1 'P = POSITION
30 F = INSTR(P,S$,T$)
35 IF F=0 THEN 60
40 C=C+1
45 PRINT LEFT$ (S$,F-1)+STRING$(LEN(T$),
   CHR$(128)) + RIGHT$(S$,LEN(S$) -F-
   LEN(T$)+1)
50 P=F+LEN(T$)
55 IF P<=LEN(S$)-LEN(T$)+1 THEN 30
60 PRINT "FOUND"; C; "OCCURRENCES"
```

The following is a sample run. However, you can input whatever text you need.

```
SEARCH TEXT? YOU SHOULD TRY TO USE YOUR TRS-
80 COLOR COMPUTER AS MUCH AS POSSIBLE.
TARGET TEST? TR
YOU SHOULD  ▒▒Y TO USE YOUR TRS-80 COLOR
COMPUTER AS MUCH AS POSSIBLE
YOU SHOULD TRY TO USE YOUR  ▒▒S-80 COLOR
COMPUTER AS MUCH AS POSSIBLE
FOUND 2 OCCURRENCES
OK
```

Here's what happens:

1. Line 15 assigns S$ (search) the value, YOU SHOULD TRY TO USE YOUR TRS-80 COLOR COMPUTER AS MUCH AS POSSIBLE.
2. Line 20 assigns T$ (target) the value of TR.
3. Line 30 tells the computer to start searching for T$ at the first position (P) in S$.
4. In Lines 45 and 55, INSTR locates T$ and then prints and blocks out T$ (CHR$(128)). It searches for the next occurrence of T$ and does the same.
5. Line 60 tells the computer to display the number of occurrences of T$ in S$.

DO-IT-YOURSELF PROGRAM 31-2

Write a program that returns the first and second occurrences of the B in ABCDEB.

The following data storage program contains a mailing list of names and addresses. This is an easy way to store information. Notice that we've saved storage space by not putting spaces between the words. Doing so makes it difficult for you to read but not for the computer to do so.

Notice also that we assign a leading asterisk (*) to zip codes so the computer doesn't confuse them with street numbers.

In this case, we're looking for the names and addresses of all individuals who live in the area specified by zip code 650—. Consequently, *650 is the *target* (A$).

```
5 CLS
10 A$ = "*650"
20 X$ = "JAMES SMITH,6550HARRISON,
       DALLASTX*75002:SUE
       SIM,RT3,GRAVIOSMO*65084:LYDIA
       LONG,3445SMITHST,ASBURYNJ*32044:
       JOHN GARDNER,BOX60EDMONTONALBERTACA"
30 Y$ = "KERRY FEWELL,456MAPLE,
       NEWORLEANS*89667: BILL
       DOLSEIN,6313E121 KANSASCITYMO*64134:
       STEVE HODGES, RT4FLORENCEME*65088
40 Z$ = "KAREN CROSS,314HURLEY
       WASHINGTONDC*10011: ASHER
       FITZGERALD,2338HARRISONFTWORTHTX
       *76101: LIZ DYLAN,BOX999NEWYORKNY
       *86866"
```

So that your computer can search X$, add this line:

```
50 PRINT INSTR(X$,A$)
```

Run the program. Your screen displays:

```
62
OK
```

This tells you the string contains a name and address you need.

What about Y$? Edit Line 50 so that the computer searches through those addresses. Does it tell you it found the needed name?

Now try Z$. Displaying a zero is your computer's way of saying, "There aren't any names you need on this list."

DO-IT-YOURSELF PROGRAM 31-3

Modify the mailing list program so that the following are true:

X$ contains two addresses that have a 650— zip.

The computer looks for every occurrence of *650, not only for the first.

Never Change Horses in Midstring (MID$)

MID$ statement gives you a powerful string editing capability by letting you replace a portion of one string with another. The syntax of MID$ is as follows:

MID$ (*oldstring,position,length*) = *newstring*

oldstring is the variable-name of the string to replace.

position is the number of the position of the first character to be changed.

length is a number of characters to replace. If you omit *length*, the computer replaces all of *oldstring*

newstring is the string that replaces the specified portion of oldstring.

Note: If *newstring* has fewer characters than *length* specifies, the computer substitutes all of *newstring*. *newstring* is always the same length as *oldstring*.

To see what we mean, run this program:

```
5 CLS
10 A$ = "KANSAS CITY, MO"
20 MID$(A$,14)="KS"
30 PRINT A$
```

Line 10 assigns A$ the value KANSAS CITY, MO. Then Line 20 tells the computer to use MID$ to replace part of the *oldstring* (A$) with KS, starting at Position 14.

Change Position 14 to 8 and run the program. The result is:

```
KANSAS CITY, MO
```

Now add the *length* option to Line 20:

```
20 MID$(A$,14,2)="KS"
```

Notice that it doesn't affect the result since *newstring and oldstring* are both two characters long. Change *length* to 1:

```
20 MID4(A$,14,1)="KS"
```

The computer replaces only one character in *oldstring*, using the first character in KS.

You'll find MID$ to be doubly effective when used with INSTR. Using the two, you can "search and destroy" text. INSTR searches; MID$ changes or "destroys." The following program illustrates this:

```
5 CLS
10 INPUT "ENTER A MONTH AND DAY (MM/DD). ";X$
20 P = INSTR(X$,"/")
30 IF P = 0 THEN 10
40 MID$(X$,P,1)= "-"
50 PRINT X$ " IS EASIER TO READ, ISN'T IT?"
```

In this program, INSTR searches for a slash (/). When it finds one, MID$ replaces it with a hyphen (-).

DO-IT-YOURSELF PROGRAM 31-4

Pretend you worked at a telephone company in the days when telephone exchanges were being switched from alpha-characters to numeric-characters. Write a program that uses MID$ to replace all alpha-exchanges with numbers. Be sure to clear enough string space or you'll get an ?OS ERROR.

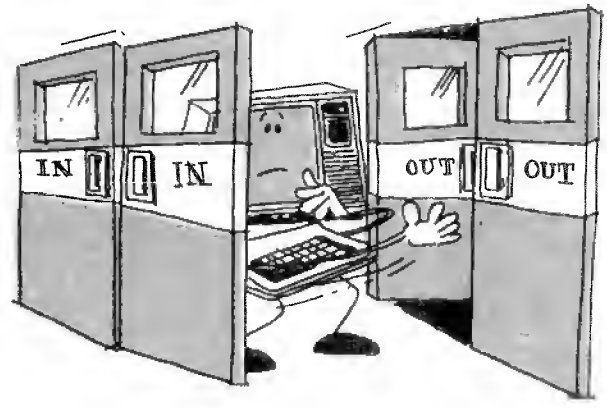
Learned in Chapter 31

BASIC WORDS	CONCEPTS
STRING$	Creating a string of characters
INSTR	Searching for a string
MID$	Replacing one string for another

Notes

IN ONE DOOR AND OUT THE OTHER

Input/output statements let you send data from the keyboard to the computer, from the computer to the TV, and from the computer to the printer. These functions are primarily used inside programs to input data and output results and messages.



A Line Drive
(LINE INPUT)

The first input/output statement is LINE INPUT. Its syntax is as follows:

LINE INPUT *"prompt" string variable*

prompt is the prompting message.

string variable is the name assigned to the line that is input from the keyboard.

LINE INPUT is similar to INPUT, except for these differences:

- When the statement executes, the computer does not display a question mark while awaiting keyboard input.
- Each LINE INPUT statement can assign a value to only one variable.
- The computer accepts commas and quotation marks as part of the string input.
- Leading blanks, rather than being ignored, become part of the string variable.

With LINE INPUT, you can input string data without worrying about accidentally including delimiters such as commas, quotation marks, and colons. The computer accepts everything. In fact, some situations require that you input commas, quotation marks, and leading blanks as part of the data.

Examples:

LINE INPUT X$ (ENTER)

lets you input X$ without displaying any prompt.

LINE INPUT "LAST NAME, FIRST NAME? ";N$ (ENTER)

displays the prompt "LAST NAME, FIRST NAME? " and inputs data. Commas do not terminate the input string. Notice that the *prompt* includes the question mark and the following space.

To understand LINE INPUT better, enter and run the following program:

```
10 CLEAR 300: CLS
20 PRINT TAB(8); "LINE INPUT STATEMENT":
   PRINT
30 PRINT: PRINT "*** ENTER TEXT ***"
40 ' *** GET STRING, THEN PRINT IT ***
50 A$ =  "" 'SET A$ TO NULL STRING
60 LINE INPUT "==> "; A$
70 IF A$ =  "" THEN END 'IF STILL NULL
   STRING, STOP!
80 PRINT A$
90 GOTO 50
```

Customized Printing
(PRINT USING)

By now you know that the more you work with your computer, the more it can work for you. For instance, maybe you want to create a table that uses numbers, but you don't want to type the plus and minus signs repeatedly.

PRINT USING makes short work of this kind of problem by enabling the computer to print strings and numbers in a "customized" format. This can be especially useful for accounting reports, checks, tables, graphs, or other output that requires a specific print format.

Here is PRINT USING's syntax:

PRINT USING *format;item-list*

format is a string expression that tells the computer the format to use in printing each item in *item-list*. It consists of "field specifiers" and other characters and is one (or one set).

item-list is the data to be formatted.

Note: PRINT USING does not automatically print leading and trailing blanks around numbers. It prints them only as you indicate in *format*.

You may use the following field specifiers as part of *format*:

#	$$	_
,	**$	^^^^
**	+	!
$		

The examples in the field specifier list are in the immediate mode but may be incorporated into a program line.

Below is an explanation of each field specifier, followed by examples of its use.

A number sign specifies the position of each digit in the number you enter. The number of number signs establishes the length of the numeric field.

If the field is larger than the number, the computer displays the unused positions to the left of the number as spaces and those to the right as zeros.

```
PRINT USING "#####"; 66.2 (ENTER)
   66
```

If the field is too small for the number, the computer displays the number with a leading % sign.

```
PRINT USING "#"; 66.2 (ENTER)
%66
```

You can place the decimal point at any field location that you established with the number sign. The computer automatically rounds off any digits to the right of the decimal point that don't fit into the field.

```
PRINT USING "#.#"; 66.25 (ENTER)
%66.3

PRINT USING "##.#"; 58.76 (ENTER)
58.8

PRINT USING "##.##
";10.2,5.3,66.789,.234 (ENTER)
10.20     5.30    66.79     0.23
```

Note: In the last example, *format* contains three spaces after the final number sign. These spaces separate the numbers when the computer displays them.

, The comma, when placed in any position between the first digit and the decimal point, displays a comma to the left of every third digit. The comma establishes an additional position in your numeric field. To avoid an overflow (indicated by a leading percent sign), place a comma at every third position in the numeric field. Overflows occur when the field isn't large enough.

```
PRINT USING "#########,"; 12345678
12,345,678

PRINT USING "#########,"; 123456789
%123,456,789

PRINT USING "###,###,###"; 123456789
123,456,789
```

****** When you place two asterisks at the beginning of the numeric field, the computer fills all unused positions to the left of the decimal with asterisks. The two asterisks establish two more positions in the numeric field.

```
PRINT USING "******"; 44.0
****44
```

$ Placing a dollar sign ahead of the numeric field causes the computer to place a dollar sign ahead of the number when displaying it. This, of course, is handy when you are working with money.

```
PRINT USING "$###.##"; 18.6735
$   18.67
```

$$ Two dollar signs placed at the beginning of the field cause the computer to display a floating dollar sign immediately preceding the first digit.

```
PRINT USING "$$##.##"; 18.6735
$18.67
```

****$** You can place this combination of symbols at the beginning of the field also. If you do, the computer fills the vacant positions to the left of the number with asterisks and places a dollar sign in the position immediately preceding the first digit.

```
PRINT USING "**$.##"; 8.333
*$8.33
```

+ When you place a plus sign at the beginning or end of the field, the computer precedes all positive numbers with a plus sign and all negative numbers with a minus sign.

```
PRINT USING "+********"; 75200
***+75200

PRINT USING "+###"; -216
-216
```

– When you place a minus sign at the end of the field, the computer follows all positive numbers with a space and precedes all negative numbers with a minus sign.

```
PRINT USING "####.#-"; -8124.420
8124.4-
```

Do you have all that memorized?

```
PRINT USING "%     %"; "BLUE'S STORE"
BLUE'S
```

To see PRINT USING in use, run the following program:

```
5 CLS
10 A$ = "**$##,######.## DOLLARS"
20 INPUT "WHAT'S YOUR FIRST NAME"; F$
30 INPUT "WHAT'S YOUR MIDDLE NAME"; M$
40 INPUT "WHAT'S YOUR LAST NAME"; L$
50 INPUT "ENTER THE AMOUNT PAYABLE"; P
60 CLS
70 PRINT "PAY TO THE ORDER OF ";
80 PRINT USING "!";F$;" ";M$;" ";
90 PRINT L$
100 PRINT: PRINT USING A$; P
110 GOTO 110
```

Line 10 defines the format, using **$ to fill the leading spaces with asterisks and placing a dollar sign directly before the first number. This format is sometimes used to protect checks from being altered.

To learn more about PRINT USING, experiment with this program:

```
5 CLS
10 INPUT
   "FORMAT";F$
20 INPUT "ITEM-
   LIST";I
30 PRINT USING
   F$;I
40 GOTO 5
```

This works fine for numeric data. For string data, change I in Lines 20 and 30 to I$.

Line 10 also sets up the numeric field using the # sign. Thus, whenever you enter a number that is smaller than the numeric field, the computer precedes the number with asterisks to fill the unused spaces. Included in Line 10 are two more field specifiers, the decimal point and the comma.

The computer displays the decimal point at only those positions specified. Because you tell the computer to include two places to the right of the decimal (for cents), the computer rounds all numbers of more than two digits to two digits. If you enter a number that has one or no digits to the right of the decimal point, the computer inserted zeros.

The exclamation marks in Line 80 tell the computer to use only the first character (the initial) of F$ (your first name) and of M$ (your middle name).

DO-IT-YOURSELF PROGRAM 32-1

Change the program so that no leading asterisks appear on the check.

DO-IT-YOURSELF PROGRAM 32-2

Write a program that creates a table showing your income and expenses on a monthly basis. Don't bother to itemize your expenses; just calculate the totals and the net result (plus or minus).

Use STRING$ to organize the table, making it flexible enough so you can use it month after month without changing the entire program.

Net results? Is this tennis or big business?

POS

POS is an input/output function that returns the current cursor position on the screen or the carriage position on the printer. Here is its syntax:

POS (*device number*)

device number is 0 (screen) or –2 (printer)

```
PRINT TAB (8) POS(0)
```

returns the number 8 at Column 8 in the current line.

Note: The leading space before "8" causes it to appear in Column 9.

One way to use POS is to disable the "wrap-around" feature on the screen or the printer. Doing this prevents words from being broken in the middle. On the other hand, it necessarily shortens the line length. Run the following program to see POS at work:

```
5 CLS
10 A$ = INKEY$
20 IF A$ = "" THEN 10
30 IF POS (0) > 22 THEN IF A$ = CHR$(32) THEN
   A$=CHR$(13)
40 PRINT A$;
50 GOTO 10
```

This program lets you use the keyboard as a typewriter (except that you can't correct mistakes unless you first disable the printer). POS watches the end of the line so no word is divided.

In Line 30, the computer checks to see if the "current" cursor position is greater than Column 22. (The screen is 32 columns wide.) If the cursor passes Column 22, the computer begins a new line the next time you press the space bar (CHR$(32)). When the computer decides to begin a new line, it does so by printing a carriage return (CHR$(13)); in effect, the computer presses (ENTER).

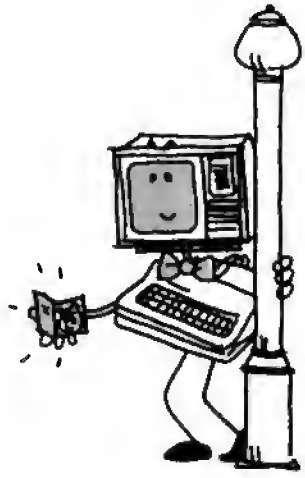
We chose to test cursor position 22 since it was 10 spaces less than the maximum screen width, 32; that gives plenty of room to complete a long word.

DO-IT-YOURSELF PROGRAM 32-3

Write a program that uses POS to space words evenly on a single line.

De-Vice Squad

Did you ever think of your video display as an "output" device and your keyboard as an "input" device?



With PRINT, PRINT USING, LINE INPUT, and POS, you can use device numbers to direct input or output. For instance, suppose you type:

```
PRINT #-2, USING "###.###";123.45678 (ENTER)
```

The screen remains "silent" while the printer prints:

```
123.456
```

You can use any of the available field specifiers with PRINT #–2, USING.

POS(–2) returns the printer's current print position (the current carriage position). Run the following program:

```
5 CLS
10 FOR I = 1 TO 10
20 PRINT #-2, "*";
30 PRINT "PRINTER POS="; POS(-2)
40 NEXT I
50 PRINT #-2," "
```

The screen shows the print carriage position as it changes. Note that the position is figured internally, not mechanically. Most printers can't print until Line 50 executes.

LINE INPUT # works similarly, with the one difference that it lets you read a "line of data" from a cassette file.

LINE INPUT # reads everything from the first character up to whichever of the following comes first:

- A carriage-return character that is not preceded by a line-feed character
- The 249th data character
- The end-of-file

Other characters encountered (quotes, commas, leading blanks, and line feed/carriage return sequences) are included in the string. For instance:

```
LINE INPUT #-1,A$
```

inputs a line of cassette file data into A$.

The following program uses LINE INPUT # to count the number of lines in any cassette-stored program that is CSAVEd in ASCII format (using the A option):

```
10 CLEAR 500
20 LINE INPUT "NAME OF DATA FILE? ";F$
30 K=0   'K IS THE COUNTER
40 OPEN "I",-1,F$
50 IF EOF (-1) THEN 100
60 LINE INPUT #-1, A$
70 K=K+1
80 PRINT A$
90 GOTO 50
100 CLOSE#-1
110 PRINT "FILE CONTAINED";K;"LINES"
```

Learned in Chapter 32

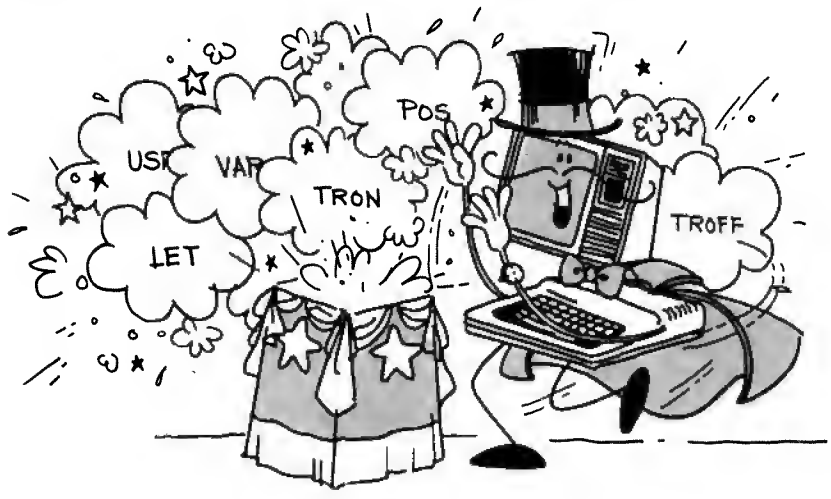
BASIC WORDS	CONCEPTS
LINE INPUT	Inputting a line from the keyboard
PRINT USING	Displaying strings and numbers in a customized format
POS	Determining the current cursor position or the current carriage position

Notes

CHAPTER 33

A LITTLE BYTE OF EVERYTHING

This chapter contains a hodge-podge of Extended Color BASIC features that don't fit neatly into categories but that, nonetheless, can be very helpful.



LET

Many versions of BASIC require that you use LET whenever you assign a value to a variable as in the statement LET X=5. Although extended Color BASIC does not require LET, you may want to use it anyway. One reason is to ensure compatibility with those versions of BASIC that do require it.

For example, these statements are the same:

```
10 LET A$ = "A#"
10 A$ = "A#"
```

TRON/TROFF Commands

TRON ("trace on") and TROFF ("trace off") are debugging aids that help you trace the execution of program statements.

TRON turns on a "tracer" that displays each line number of the program as it is executed. The numbers appear enclosed in brackets. TROFF turns off the tracer.

Examples:

```
TRON (ENTER)
TROFF (ENTER)
```

Trace the execution of the "Lines" program. Type TRON (ENTER). Then run the program:

```
5 PCLS
10 PMODE 3,1
20 SCREEN 1,1
30 LINE (0,0)-(255,191),PSET
```

The computer displays:

```
(5) (10) (20) (30)
OK
```

This display indicates that the program first executed Line 5, then 10, 20, and finally 30. Remember to type TROFF (ENTER) to turn off the tracer.

Time After Timer . . .
(TIMER)

Your computer also has a built-in "timer" that measures time in sixtieths of a second (approximately). The moment you power-up the computer, the timer begins counting at zero. When it counts to 65535 (approximately 18 minutes later), the timer starts over at zero. It pauses during cassette and printer operations.

At any instant, you can see the count of the timer by using the TIMER function. Type:

```
PRINT TIMER (ENTER)
```

The TIMER function displays a value from 0 to 65535.

You can also reset the timer to any specified time by typing:

```
TIMER = number (ENTER)
```

number is in the range 0 to 65535.

To see TIMER (and PRINT @ USING, another "new" function), run the following program called "Math Quiz." It presents you with a math problem. When you press (A), (B), (C), or (D), the computer tells you whether the answer is right or wrong. Then the computer uses the timer to tell you the time you took to answer (using TIMER).

```
10 DIM CH(3),L$(3) 'CH(#)=CHOICES,
   L$=ANSWER FORMATS
20 LL=10:UL=20 'LOWER LIMIT AND UPPER LIMIT
   FOR H AND V
30 NV=UL-LL+1
40 P$="WHAT'S ### + ### ?" 'QUESTION FORMAT
50 FOR I = 0 TO 3 'INITIALIZE CH( )
60 L$(I)=CHR$(I+65)+") ###"
70 NEXT I
80 CLS
90 X=INT(RND(NV(+LL-.5) 'GET RANDOM X
   BETWEEN LL AND UL
100 Y=INT(RND(NV)+LL-.5) 'GET RANDOM Y
    BETWEEN LL AND UL
110 R=INT(X+Y+.5) 'CORRECT ANSWER
```

```
130 FOR I = 0 TO 3 'GET MULT. CHOICES
140 CH(I)=INT(RND(NV)+LL-.5)
150 NEXT I
160 RC=RND(4)-1 'MAKE ONE CHOICE RIGHT
170 CH(RC)=R
180 PRINT @ 32, USING P$;X,Y
    'DISPLAY PROBLEM
190 FOR LN=3 TO 6
200 PRINT @ LN * 32+10,USING L$(LN-3);CH
    (LN-3)
210 NEXT LN
220 TIMER = 0
230 A$=" " 'CLEAR KEYBOARD
240 A$=INKEY$: IF A$="" THEN 240
250 SV=TIMER 'IF KEY PRESSED, SAVE TIMER
    CONTENTS
260 IF A$<"A" OR A$>"D" THEN 240 'INVALID
    KEY-GO BACK
265 PRINT @ 8 * 32+10,A$
270 K=ASC(A$)-65
280 IF CH(K)=R THEN PRINT "RIGHT!": GOTO 300
290 PRINT "WRONG! ANSWER IS "; R
300 PRINT "YOU TOOK"; SV/60; "SECONDS"
310 INPUT "PRESS <ENTER> FOR NEXT PROBLEM";
    EN
320 GOTO 80
```

Through trial and error, change the upper and lower limits (Line 20) for *h* and *v*. Make the program perform a mathematical operation other than addition or have the computer keep score, based on your time. Add 5 seconds for each incorrect answer.

Hexadecimal and Octal Constants

Extended Color BASIC lets you use both hexadecimal and octal constants.

Hexadecimal numbers are quantities represented in Base 16 notation, composed of the numerals 0 to 9 and the "numerals" A to F. Hexadecimal constants must be in the range 0 to FFFF, corresponding to the decimal range 0 to 65535.

To indicate that a number is an octal constant, precede it with the symbol &H, as shown here:

&HA010 &HFE &HD1 &HC &H4000

Octal numbers are quantities represented in Base 8 notation, composed of the numerals 0 to 7. Octal constants must be in the range 0 to 177777. The computer stores them as two-byte integers that correspond to the decimal range 0 to 65535.

To indicate that a number is an octal constant, precede it with the symbol &O or &, as shown here:

&O70 &O44 U1777 &7170 &17 &O1234

The use of "hex" and octal constants is convenient in programs that reference memory locations and contents. For further information, read a book on machine-language programming.

HEX$

To convert a number from decimal to hexadecimal, use HEX$. The syntax is as follows:

HEX$ (*number*)

number is a decimal number of variable from 0 to 65535.

For example, the following program displays the hexadecimal value of any decimal number smaller than 65536. It returns a string that represents a hex value.

```
5 CLS
10 INPUT "IF A NUMBER'S DECIMAL VALUE IS";
   DEC
20 PRINT "ITS HEXADECIMAL VALUE IS "
   HEX$(DEC)
```

Learned in Chapter 33

BASIC WORDS	CONCEPTS
LET	Using LET to make programs compatible with other versions of BASIC
TRON, TROFF	Using the tracer to follow the execution of program statements
TIMER	Keeping track of and changing the time in a program
HEX$	Converting a number from decimal to hexadecimal

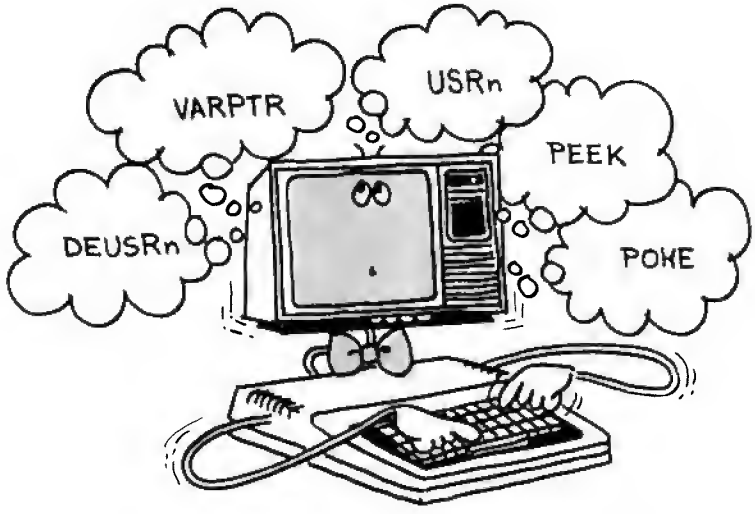
Notes

Chapter 34

USING MACHINE-LANGUAGE SUBROUTINES

"Machine-language" (ML) is the low-level language that your computer uses internally. It consists of microprocessor instructions. ML subroutines are useful for special applications simply because they can do things much faster than BASIC.

Writing such routines requires familiarity with assembly-language programming and with the microprocessor's instruction set. For more information, see *6809 Assembly Language Programming*, Lance Leventhal, Osborne/McGraw Hill, 1981.



This section follows the step-by-step approach for using ML subroutines:

1. Protecting Memory
2. Storing the ML Subroutine in Memory
3. Telling BASIC Where the Subroutine Is
4. Calling the Subroutine
5. Returning to BASIC

We present a sample BASIC program that performs all five steps. You may type in the BASIC program lines as they are given, but don't try to run the program until you've read all the steps.

Our ML subroutine is simple. It gets a character from the keyboard. Then it returns the ASCII code for this character to the BASIC program. An assembly-language listing of this routine is later in this section.

Our ML subroutine has a few features not available with BASIC's INKEY$ or INPUT statements. First, it returns any key code, including the one for (BREAK). Second, it lets you key in control codes A-Z (CTRL-A through CTRL-Z).

To key in a control character, press (↑), release it, then press any key from (A) to (Z). The control codes generated range from 1 to 26.

STEP 1. PROTECTING MEMORY

With the CLEAR statement, you can reserve a section of memory for storing your ML subroutine. The first CLEAR parameter sets the string space; the second sets the memory protection address. For example:

```
5 CLEAR 25, 12000
```

sets the string space to 25 bytes and reserves memory addresses from 12000 to the end of memory (see the Memory Map). You can now safely store your ML subroutine in this area.

STEP 2. STORING THE ML SUBROUTINE IN MEMORY

You can load an ML subroutine from tape (via CLOADM), or you can poke it into memory (using the BASIC POKE statement). In our example, we'll store the individual machine codes in DATA statements, then read and poke each code into the correct memory address. The codes are in the ML subroutine's assembly listing, shown later in this section.

```
20 FOR I = 1 TO 28
30 READ B: POKE 12000 + I, B
40 NEXT I
50 DATA 173, 159, 160, 0
60 DATA 39, 250, 129, 10, 38, 12
70 DATA 173, 159, 160, 0, 39, 250
75 DATA 129, 65, 45, 2
80 DATA 128, 64, 31, 137, 79
90 DATA 126, 180, 244
```

STEP 3. TELLING BASIC WHERE THE SUBROUTINE IS

Before you can use the ML subroutine, you have to tell BASIC where the routine starts. Do this with the DEFUSR statement, which has this format:

DEFUSR*n* **=** ***address*** tells where, in memory, an ML subroutine starts

n is the number of the ML subroutine (0-9).
address is the first address in memory where the ML subroutine is stored.

In this example, the ML subroutine (which we'll call ML Subroutine 1) is stored in memory starting at Address 12000. To tell this to BASIC, use this statement:

```
10 DEFUSR1 12000
```

STEP 4. CALLING THE SUBROUTINE

To "call" the ML subroutine, use the USR function with this format:

dummy variable **=** **USR*n*(*argument*)** calls an ML subroutine

n is the number of the ML subroutine (0-9).
argument is a value you want to pass to the ML subroutine.
dummy variable is a variable you can use to store the data returned by USR.

For example:

```
110 A = USR1(0)
```

calls ML Subroutine 1 and passes it Argument 0. In this example, 0 is a "dummy argument." The ML subroutine won't use it. (The purpose of Variable A is explained in the next step.)

STEP 5. RETURNING TO BASIC

If you want to return a specific integer value to BASIC, as we do in this example, your ML subroutine must: (1) load the integer into Register D, (2) end by calling GIVABF, a special ROM subroutine. GIVABF causes your BASIC program's USR function to "return," replaced by the integer you stored in Register D.

The address of GIVABF is Hexadecimal B4F4 or Decimal 46324. However, if you have Advanced Color BASIC or Extended Color BASIC Version 1.2 or later, this address may have been changed.

In this example, our ML subroutine loads the key you press into Register D and then calls GIVABF. This causes USR to return replaced by the key you press. Since Variable A equals the value USR returns, Variable A equals the key you press.

If you don't want to return a specific value to BASIC, end the subroutine with an RTS instruction. USR "returns" your original *dummy argument* (0).

The BASIC Program

This is the entire program with the ML subroutine poked into memory. Type it in carefully; then run it.

If you have a Deluxe Color Computer, use the **CTRL** *key, rather than* ↑.

Each time you press a key, control returns to BASIC with the ASCII code for that key. Try pressing **BREAK**. You'll get the code for **BREAK** 3. The BASIC program ends when you press **ENTER** or ↑ **M**.

To get any of the codes 1 through 26, press ↑, release it, then press a key from **A** to **Z**.

```
5 CLEAR 25, 12000 'RESERVE MEMORY
10 DEFUSR1=120001S CLS
20 FOR I = 1 TO 28 "STORE EACH BYTE OF OBJECT
   CODE
30 READ B: POKE 12000 + I, B
40 NEXT I
45 'HERE IS THE OBJECT CODE
50 DATA 173, 159, 160, 0
60 DATA 39, 250, 129, 10, 38, 12
70 DATA 173, 159, 160, 0, 39, 250
75 DATA 129, 65, 45, 2
80 DATA 128, 64, 31, 137, 79
90 DATA 126, 180, 244
99 'TELL BASIC WHERE THE ROUTINE IS
100 POKE 275, 15: POKE 276, 211
110 A = USR1(0) 'CALL THE SUBROUTINE AND GIVE
    RESULT TO A
115 IF A = 13 THEN END
120 PRINT "CODE ="; A
130 GOTO 110
```

For a variation in the program, change line 120 to:

```
120 PRINT CHR$(A);    'DISPLAY THE CHARACTER
```

Most control keys (↑ followed by a key) have no effect when printed. Try ↑ **H**, though, and you see the cursor backspace.

ML Subroutine Listing

This is the assembly-language listing of our ML subroutine example. To use it, you must have an assembler, such as EDTASM (Catalog #26-3250) or Disk EDTASM (Catalog #26-3254). You can't use this assembly-language listing from BASIC.

Assembly language is not meaningful to the computer. It is a set of memory aids and symbols we use for convenience. Assembly language must be translated, or "assembled," into machine code, which the computer understands. In the listing above, the machine code is given in hexadecimal form. We converted it to decimal numbers for our BASIC program.

```
Hexadecimal
Object Code          Source   Code          Comments

AD 9F A0 00          LOOP1    JSR (POLCAT)  ;POLL FOR A KEY
27 FA                         BEQ  LOOP1    ;IF NONE, RETRY
81 0A                         CMPA #10      ;CTRL KEY (DN
                                             ARW)?
26 0C                         BNE  OUT      ;NO, SO EXIT
AD 9F A0 00          LOOP2    JSR  (POLCAT) ;YES. SO GET NEXT
                                             KEY
27 FA                         BEQ  LOOP2    ;IF NONE, RETRY
81 20                         CMPA #65      ;IS IT A - Z?
2D 02                         BLT  OUT      ;IF < A, EXIT
80 40                         SUBA #64      ;CONVERT TO CTRL
                                             A/Z
1F 89                OUT      TFR  A,B      ;GET RETURN BYTE
                                             READY
4F                            CLRA          ;ZERO MSB
7E B4 F4                      JMP  GIVABF   ;RETURN VALUE TO
                                             BASIC

POLCAT EQU  40960
GIVABF EQU  46324
```

Passing Values to an ML Subroutine

USING THE INTCNV ROUTINE

The address of INTCNV is Hexadecimal B3ED. However, if you have Advanced Color BASIC or Extended Color BASIC Version 1.2 or later, this address may have been changed.

If you want to pass an integer to your ML subroutine, use the integer as the "argument" in your USR function. For example:

```
A=USR1(5)
```

calls Machine Code Program 1 and passes the argument 5 to it. You can then call the INTCNV routine, which gets the integer and stores it in Register D.

USING THE VARPTR FUNCTION

Another way to pass an argument to your ML Subroutine is to pass a "pointer" to the address where a variable's value is stored. You can do this with the VARPTR function:

VARPTR *variable* returns a pointer to where the variable's value is stored

For example:

```
A=USR1(VARPTR(B))
```

calls ML Subroutine 1 and passes a pointer to Variable B's address. The pointer is stored in Register X. Your ML subroutine needs to know whether the variable is string or numeric.

If the variable is string, your ML subroutine can find the string's 5-byte descriptor in Register X. This descriptor tells where the string is:

Byte 1 = the length of the string (in characters)
Byte 2 = reserved for the computer's use
Bytes 3 and 4 = address of the first byte in the string
Byte 5 = reserved for the computer's use

If the variable is numeric, your program can find the address of the number's floating point value in Register X. This floating point value has this format:

Byte 1 = the exponent of the mantissa
Byte 2 = the mantissa's most significant byte (MSB)
Byte 3 = the mantissa's next MSB
Byte 4 = the mantissa's next MSB
Byte 5 = the mantissa's least significant byte (LSB)

The exponent is a signed 8-bit integer with 128 decimals added to it. An exponent of 0 means the number is 0, in which case the mantissa is insignificant. The exponent's most significant bit stores the exponent's sign: 0 if positive, 1 if negative.

The mantissa is stored in normalized form with the most significant bit of the mantissa's MSB assumed to be 1. This bit can indicate the mantissa's sign: 0 if positive, 1 if negative.

You may want to use VARPTR to pass an array variable's pointer to an ML subroutine. For example:

```
A=USR1(VARPTR(B(5)))
```

calls ML Subroutine 1 and passes a pointer to Array B's Element 5.

Your ML subroutine can find the elements' values in memory as follows (from low to high memory):

Value of first element of last dimension
Value of last element of last dimension
Value of first element of first dimension
Value of last element of first dimension

Each element is five bytes long.

Returning Values to BASIC

USR always returns at least one value to BASIC. This value is the argument you originally pass to the ML subroutine, unless your ML subroutine changes or modifies it, as described below.

USING GIVABF TO RETURN AN INTEGER

To return a specified integer to BASIC, you can have your ML subroutine load the integer into Register D and call GIVABF, as demonstrated earlier.

MODIFYING BASIC VARIABLES

You can return any specified value to BASIC by having your ML subroutine modify a BASIC variable's value. For example, assume you call an ML subroutine with this statement:

```
A$=(USR1(VARPTR(B$))
```

You can have your ML subroutine modify B$'s value and then end the routine with an RTS instruction. This causes USR to return with B$'s modified value.

If your ML subroutine modifies a string variable, be careful of the following:

Although you can change a string descriptor's length byte to "shorten" a string, you cannot "lengthen" a string. If you don't know what size string your ML subroutine will return, reserve 255 bytes (the maximum size) for the string's value before passing it to the ML subroutine. For example:

```
B$=STRING$(255)
A$ = USR0(VARPTR(B$))
```

passes a pointer to a 255-character string of blank spaces to the USR function. The ML subroutine can then put a string of up to 255 characters into the memory pointed to by B$ or, if necessary, shorten the string's length byte.

You can modify the starting address of a string by changing the 2-byte pointer in the string descriptor. When you do this, though, we recommend the new starting address be an address included in the original string.

You can swap the starting addresses of two strings. This may be useful for sorting strings. If you do this, though, be careful not to "intersect" two strings.

If your ML subroutine modifies a variable that already points to a string literal, this will change your BASIC program. For example, assume you have this statement in your BASIC program:

```
B$ = "ABC"
```

If your ML subroutine modifies B$, your BASIC program is changed. To avoid this problem, add a null string ("") to any string literal that your ML subroutine will modify. For example:

```
B$ = "ABC" + ""
```

The null string forces BASIC to copy the string into string space, where your ML subroutine can safely modify it.

Using Stack Space

An ML subroutine, called by USR, that requires more than 30 bytes of stack storage must provide its own stack area. Save BASIC's stack pointer upon entry to the USR function, setting up a new stack pointer and restoring BASIC's stack pointer prior to returning to BASIC. The values of the A, B, X, and CC registers need not be preserved by USR.

Notes

SECTION V

ODDS AND ENDS

SUGGESTED ANSWERS TO DO-IT-YOURSELF PROGRAMS

Do-It-Yourself Program 4-4

Sounding tones from bottom of range to top and back to bottom:

```
10 FOR X = 1 TO 255
20 SOUND X,1
30 NEXT X
40 FOR X = 255 TO 1 STEP -1
50 SOUND X,1
60 NEXT X
```

Do-It-Yourself Program 5-2

Lines added to clock program:

```
92 FOR T = 200 TO 210 STEP 5
94 SOUND T,1
95 NEXT T
97 FOR T = 210 TO 200 STEP -5
98 SOUND T,1
99 NEXT T
```

Do-It-Yourself Program 5-3

```
10 FOR C = 0 TO 8
20 CLS(C)
30 FOR X = 1 TO 460
40 NEXT X
50 NEXT C
```

Do-It-Yourself Program 7-2

```
5 FOR N = 1 TO 10
10 PRINT "CHOOSE YOUR CHAMBER(1-10)"
20 INPUT X
30 IF X = RND(10) THEN 100
40 SOUND 200, 1
50 PRINT "--CLICK--"
60 NEXT N
65 CLS
70 PRINT @ 230, "CONGRATULATIONS!!!"
80 PRINT @ 265, "YOU MANAGED"
90 PRINT @ 296, "TO STAY ALIVE"
95 END
100 FOR T = 133 TO 1 STEP -5
110 PRINT "BANG!!!!!"
120 SOUND T, 1
130 NEXT T
140 CLS
150 PRINT @ 230, "SORRY, YOU'RE DEAD"
160 SOUND 1, 50
170 PRINT @ 290, "NEXT VICTIM PLEASE"
```

Do-It-Yourself Program 7-3

```
10 CLS
20 A = RND(6)
30 B = RND(6)
40 R = A + B
50 PRINT @ 200, A
60 PRINT @ 214, B
70 PRINT @ 394, "YOU ROLLED A" R
80 IF R = 2 THEN 600
90 IF R = 3 THEN 600
100 IF R = 12 THEN 600
110 IF R = 7 THEN 500
120 IF R = 11 THEN 500
130 FOR X = 1 TO 800
140 NEXT X
150 CLS
160 PRINT @ 195, "ROLL ANOTHER" R "AND YOU
    WIN"
170 PRINT @ 262, "ROLL A 7 AND YOU LOSE"
180 PRINT @ 420, "PRESS <ENTER> WHEN READY"
185 PRINT @ 456, "FOR YOUR NEXT ROLL"
190 INPUT A$
200 X = RND(6)
210 Y = RND(6)
220 Z = X + Y
225 CLS
230 PRINT @ 200, X
240 PRINT @ 214, Y
250 PRINT @ 394, "YOU ROLLED A" Z
260 IF Z = R THEN 500
270 IF Z = 7 THEN 600
280 GOTO 180
500 FOR X = 1 TO 1000
510 NEXT X
515 CLS
520 PRINT @ 230, "YOU'RE THE WINNER"
530 PRINT @ 294, "CONGRATULATIONS!!!"
540 GOTO 630
600 FOR X = 1 TO 1000
610 NEXT X
615 CLS
620 PRINT @ 264, "SORRY, YOU LOSE"
630 PRINT @ 458, "GAME'S OVER"
```

Do-It-Yourself Program 8-2

```
5 CLS
6 PRINT @ 230, "YOUR NAME";
8 INPUT N$
10 CLS
15 T = T + 1
20 X = RND(100)
30 Y = RND(100)
```

```
40 PRINT @ 228, "WHAT IS" X "+" Y;
45 INPUT A
50 IF A = X + Y THEN 82
60 PRINT @ 326, "THE ANSWER IS" X + Y
70 PRINT @ 385, "BETTER LUCK NEXT TIME," N$
80 GOTO 100
82 CLS(7)
83 FOR M = 1 TO 4
84 SOUND 175, 1
85 SOUND 200, 1
86 NEXT M
87 CLS
90 PRINT @ 232, "CORRECT," N$ "!!!"
95 C = C + 1
97 PRINT @ 299, "THAT IS"
98 PRINT @ 322, C "OUT OF" T "CORRECT
   ANSWERS"
99 PRINT @ 362, C/T*100 "% CORRECT":IF T=10 THEN
END
100 PRINT @ 420, "PRESS <ENTER> WHEN READY"
102 PRINT @ 458, "FOR ANOTHER"
105 INPUT A$
110 GOTO 10
```

Do-It-Yourself Program 10-1

```
5 CLS
7 PRINT @ 38, "TABLE OF SQUARES"
8 PRINT
10 P = 2
20 FOR N = 2 TO 10
25 GOSUB 2000
30 PRINT N "*" N "=" E,
40 NEXT N
50 END
2000 REM FORMULA FOR RAISING A NUMBER TO A
     POWER
2010 E = 1
2020 FOR X = 1 TO P
2030 E = E * N
2040 NEXT X
2050 IF P = 0 THEN E = 1
2060 RETURN
```

Do-It-Yourself Challenger Program (Chap. 11)

```
10 PRINT "TYPE A SENTENCE :"
15 INPUT S$
20 PRINT "TYPE A PHRASE TO DELETE"
23 INPUT D$
25 L = LEN(D$)
30 PRINT "TYPE A REPLACEMENT PHRASE"
35 INPUT R$
40 FOR X = 1 TO LEN(S$)
50 IF MID$(S$,X,L) = D$ THEN 100
```

```
60 NEXT X
70 PRINT D$ "-- IS NOT IN YOUR SENTENCE"
80 GOTO 20
100 E = X - 1 + LEN(D$)
110 NS$ = LEFT$(S$,X-1) + R$ +
    RIGHT$(S$,LEN(S$) - E)
120 PRINT "NEW SENTENCE IS :"
130 PRINT NS$
```

Do-It-Yourself Program 14-2

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 X = RND(256)-1
40 Y = RND(192)-1
50 C = RND(9)-1
60 PSET(X,Y,C)
70 GOTO 30
```

Do-It-Yourself Program 15-1

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
25 LINE (0,0)-(255,191),PSET
30 LINE (0,191)-(255,0),PSET
35 LINE (10,10)-(255,191),PSET,B
40 GOTO 40
```

Do-It-Yourself Program 15-2

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 LINE (72,168)-(200,72),
   PSET,B                    'FRAME
40 LINE (72,72)-(136,36),
   PSET                      'ROOF
45 LINE (200,72)-(136,36),
   PSET                      'ROOF
50 LINE (120,168)-(152,100),
   PSET,B                    'DOOR
55 LINE (152,60)-(168,36),
   PSET,BF                   'CHIMNEY
60 LINE (165,128)-(191,100),
   PSET,B                    'WINDOW
65 LINE (178,128)-(178,100),
   PSET                      'WINDOW PART
70 LINE (165,114)-(191,114),
   PSET                      'WINDOW PART
75 LINE (85,128)-(111,100),
   PSET,B                    'WINDOW
```

```
80 LINE (85,114)-(111,114),
   PSET                        'WINDOW PART
85 LINE (98,100)-(98,128),
   PSET                        'WINDOW PART
90 GOTO 90
```

Do-It-Yourself Program 15-3

```
5 PMODE 1,1
10 PCLS
20 SCREEN 1,1
30 Y=0
40 FOR X = 0 TO 200 STEP 10
50 OY = Y
60 Y = 30-OY
70 LINE (X,100-Y)-(X+10,100-OY),PSET
80 NEXT
90 GOTO 90
```

Do-It-Yourself Program 16-1

```
1 Y = -1
5 CLS
10 PRINT @ 193,"DO YOU WANT TO SEE A SQUARE?"
20 FOR X = 1 TO 1000: NEXT X
30 PMODE 1,1
35 PCLS
40 SCREEN 1,Y+1
60 LINE (75,150)-(150,75),PSET,B
70 FOR X = 1 TO 1000: NEXT X
75 Y = -Y
80 GOTO 5
```

Do-It-Yourself Program 18-1

Make the following changes:

```
22 PCOPY 4 TO 3
32 PCOPY 3 TO 2
42 PCOPY 2 TO 1
```

Delete Lines 11, 21, and 31.

Do-It-Yourself Program 18-2

```
10 PCLEAR 8
20 PMODE 4,1
25 PCLS
30 SCREEN 1,1
40 LINE (0,0)-(255,191),PSET
45 FOR Y = 1 TO 20: NEXT Y
50 PMODE 4,2
```

```
55 SCREEN 1,0
60 LINE (0,0)-(255,191),PSET
65 FOR Z = 1 TO 20: NEXT Z
70 PMODE 0,3
75 SCREEN 1,1
80 LINE (0,0)-(255,191),PSET
85 FOR A = 1 TO 20: NEXT A
90 PMODE 1,4
95 SCREEN 1,1
96 PCLS
100 LINE (0,0)-(255,191),PSET
105 FOR R = 1 TO 20: NEXT R
110 GOTO 20
```

Do-It-Yourself Program 19-1

```
10 PMODE 4,1
20 PCLS
30 SCREEN 1,0
40 FOR RADIUS = 1 TO 100 STEP 10
50 CIRCLE (128, 96),RADIUS
60 NEXT RADIUS
70 GOTO 70
```

Do-It-Yourself Program 19-3

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,0
30 CIRCLE (200,40),30,,1,.13,.63
40 CIRCLE (230,10),52,,1,.29,.48
50 GOTO 50
```

Do-It-Yourself Program 19-4

```
5 PMODE 1,1
10 SCREEN 1,0
15 PCLS 3
20 COLOR 1,0
25 CIRCLE (200,40),30,,1,.13,.63 'MOON
30 CIRCLE (230,10),52,,1,.29,.48 'MOON
35 LINE (100,185)-(180,125),PSET,B
   'HOUSE FRAME
40 LINE -(140,85),PSET        'ROOF
45 LINE -(100,125),PSET       'ROOF
55 LINE (110,160)-(125,130),PSET,B
   'WINDOW
60 LINE (155,160)-(170,130),PSET,B
   'WINDOW
70 LINE (130,130)-(149,185),PSET,B
   'DOOR
75 PSET (134,157,1)       'DOOR KNOB
80 LINE (160,105)-(160,90),PSET   'CHIMNEY
```

```
85 LINE -(175,90),PSET     'CHIMNEY
90 LINE -(175,115),PSET     'CHIMNEY
100 ' SMOKE STARTS HERE
105 X=167:Y=89    'CIRCLE CENTERPOINT
110 SP=0: EP=0     'CIRCLE START AND END
    POINT
115 FOR R = 1 TO 50 STEP .05   'CIRCLE RADIUS
120 EP=EP+.02: IF EP > 1 THEN EP = 0
125 CIRCLE (X+R, Y-R),R,4,1,SP,EP   'SMOKE
130 NEXT R
200 GOTO 200
```

Do-It-Yourself Program 20-1

Delete Line 40 and add Line 65:

```
65 PAINT (150,100),8,8
```

Do-It-Yourself Program 20-3

```
5 PMODE 1,1
10 PCLS
15 SCREEN 1,0
20 PCLS 3
25 COLOR 1,0
30 CIRCLE (200,30),15
35 PAINT (200,30),2,1
40 LINE (100,185)-(180,125),PSET,B
45 LINE -(140,90),PSET
50 LINE -(100,125),PSET
55 PAINT (135,115),4,1
60 LINE (110,160)-(125,130),PSET,B
65 LINE (155,160)-(170,130),PSET,B
70 PSET (134,157,1)
75 PAINT (120,180),0,1
80 LINE (130,130)-(149,185),PSET,B
85 LINE (101,135)-(41,185),PSET,B
90 LINE (91,140)-(51,185),PSET,B
95 PAINT (55,138),0,1
100 PAINT (89,183),4,1
105 FOR X = 1 TO 500: NEXT X
110 PAINT (89,183),2,1
115 FOR X = 1 TO 500: NEXT X
120 PAINT (89,155),4,1
140 GOTO 110
```

Do-It-Yourself Program 21-1

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,0
```

```
30 DRAW "BM68,116;E20;BE20;E20;F20;BF20;
   F20;L40;BL40;L40;BU40;R40;BR40;
   R40;G20;BG20;G20;H20;BH20;H20;BM128,96;
   NU40;ND40;NE20;NF20;NG20;NH20;NL40;R40"
40 GOTO 40
```

The star you created probably isn't as fancy as this one because you haven't been introduced to B or N yet. But don't worry; you will be before the end of the chapter.

Do-It-Yourself Program 21-2

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,1
25 DRAW "BM40,80;U40;R40;D40;L40"
30 DRAW "BM+20,20;U40;R40;D40;L40"
40 LINE (60,100)-(40,80),PSET
50 LINE (60,60)-(40,40),PSET
60 LINE (100,60)-(80,40),PSET
70 LINE (100,100)-(80,80),PSET
80 GOTO 80
```

Do-It-Yourself Program 21-3

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,1
25 DRAW "BM50,50L30D30R30D30L30"
30 DRAW "BM90,50D60R30U60"
40 DRAW "BM160,50D60R30BU60L30D30R30"
50 GOTO 50
```

Do-It-Yourself Program 21-4

```
5 PMODE 4,1
10 PCLS
20 SCREEN 1,0
30 DRAW "BM98,96;NU80;NE56;NR80;NF56;
   ND80;NG56;NL80;NH56"
40 CIRCLE (98,96),80,1,1,,125,1
50 CIRCLE (135,110),80,1,1,1,,125
60 LINE (135,110)-(190,167),PSET
70 LINE (135,110)-(213,110),PSET
80 GOTO 80
```

Do-It-Yourself Program 21-5

```
1 CLEAR 2500
5 DIM AZ$(25)
6 FOR LE = 0 TO 25
10 READ AZ$(LE)
15 NEXT LE
```

```
20 NC$="BR4BU7"   'NEXT CHARACTER
25 NL$="BD4"   'NEXT LINE
30 BS$="BL9"   'BACKSPACE
35 HM$="BM0,10"   'HOME POSITION
100 CW=6: CH=8   'SIZE OF CELL
110 R1=7: R24=191   'ROW POSITION
120 C1=8: C42=247   'COLUMN POS
125 CC=1: CL=1   'CURRENT ROW/COL
200 PMODE 4,1
210 PCLS
220 SCREEN 1,0
225 DRAW "S4"
230 DRAW HM$
250 A$=INKEY$: IF A$=" " THEN 250
260 IF "A">A$ OR "Z" < A$ THEN 250
262 CC=CC+ 1
265 IF CC>27 THEN DRAW NL$: FOR I = 1 TO 27:
    DRAW BS$: NEXT I:CC=1: GOTO 270
269 DRAW NC$
270 DRAW AZ$(ASC(A$)-65)
290 GOTO 250
1000 ' A
1010 DATA BD1D6U4NR5U2E1R3F1D6
1020 ' B
1030 DATA ND7R4F1D1G1NL4F1D2G1NL4BR1
1040 ' C
1050 DATA BD1D5F1R3E1U1BU3U1H1L3G1BD6BR5
1060 ' D
1070 DATA D7R4E1U5H1L4BD7BR5
1080 ' E
1090 DATA NR5D3NR4D4R5
1100 ' F
1110 DATA NR5D3NR4D4BR5
1120 ' G
1130 DATA BD1D5F1R3E1U2NL2BU2U1H1L3G1BD6BR5
1140 ' H
1150 DATA D7U4R5NU3D4
1160 ' I
1170 DATA R4L2D7L2R4BR1
1180 ' J
1190 DATA BD5D1F1R3E1U6BD7
1200 ' K
1210 DATA D7U4R3E2NU1G2F2D2
1220 ' L
1230 DATA D7R5
1240 ' M
1250 DATA ND7R2ND7R2D7BR1
1260 ' N
1270 DATA D1ND6E1R3F1D6
1280 ' O
1290 DATA BD1D5F1R3E1U5H1L3G1BD6BR5
1300 ' P
1310 DATA ND7R4F1D2G1L4BD3BR5
1320 ' Q
1330 DATA BD1D5F1R3E1U5H1L3G1D4BR3F2
1340 ' R
1350 DATA ND7R4F1D1G1NL4F1D3
```

```
1360 ' S
1370 DATA BD1D1F1R3F1D2G1L3H1BU5E1R3F1BD6
1380 ' T
1390 DATA R4L2D7BR3
1400 ' U
1410 DATA D6F1R3E1U6BD7
1420 ' V
1430 DATA D5F2E2U5BD7BR1
1440 ' W
1450 DATA D7R2NU6R2U7BD7BR1
1460 ' X
1470 DATA D1F5D1BL5U1E5U1BD7
1480 ' Y
1490 DATA D2F2ND3E2U2BD7BR1
1500 ' Z
1510 DATA R5D1G5D1R5
```

Do-It-Yourself Program 21-6

```
5 PMODE 3,1
10 PCLS
15 SCREEN 1,0
20 DRAW "BM50,170;U80;NG30;E80;F80;NF30;
   D80;L50;U70;L50;D70;L60"
25 LINE (50,170)-(170,170),PSET
30 LINE (110,170)-(160,170),PSET
35 FOR X = 1 TO 500: NEXT X
40 LINE (100,170)-(160,170),PRESET
45 LINE (120,180)-(120,110),PSET
50 LINE (160,100)-(125,110),PSET
55 LINE (160,170)-(125,180),PSET
60 LINE (120,180)-(120,110),PRESET
65 LINE (160,100)-(125,110),PRESET
70 LINE (160,170)-(125,180),PRESET
75 DRAW "BM110,170;BU70;BR50;G25;D70;E25"
80 CIRCLE (130,125),10,,1,,135,,9
85 DRAW "BM130,130;D15;D15;G10;E10;U15;L10"
90 LINE (120,145)-(120,135),PSET
91 FOR X = 1 TO 60: NEXT X
95 LINE (120,145)-(120,135),PRESET
96 FOR X = 1 TO 120: NEXT X
100 LINE (120,145)-(110,145),PSET
101 FOR X = 1 TO 60: NEXT X
105 LINE (120,145)-(110,145),PRESET
106 FOR X = 1 TO 60: NEXT X
110 LINE (120,145)-(120,135),PSET
120 FOR X = 1 TO 120: NEXT X
121 CIRCLE (130,125),10,1
122 DRAW "BM130,130;C1;D30;G10;E10;U15;L10"
125 DRAW "BM110,170;BU70;BR50;C1;
    G25;D70;E25;"
130 COLOR 4,1
135 LINE (120,180)-(120,110),PSET
140 LINE (160,100)-(125,110),PSET
145 LINE (160,170)-(125,180),PSET
```

```
150 LINE (120,180)-(120,110),PRESET
155 LINE (160,100)-(125,110),PRESET
160 LINE (160,170)-(125,180),PRESET
165 LINE (110,170)-(160,170),PSET
170 FOR X = 1 TO 500: NEXT X
175 GOTO 20
```

Do-It-Yourself Program 22-1

```
5 PCLEAR 4
10 PMODE 4,1
15 PCLS
20 SCREEN 1,1
25 DIM V(35,35)
30 X=10: Y=10
35 DRAW "BM10,10; S2; H10;R15;F10;R20; F10;
   G10;L20;G10;L15;E10;U20;D4;NL8;D4;NL12;
   D4NL16; D4;NL12;D4;NL8"
40 GET (X-X,Y-Y)-(X*3.5,Y*3.5),V,G
45 A$=INKEY$: IF A$=" " THEN 45  'PRESS ANY
   KEY TO START
50 PCLS
55 FOR A = 10 TO 200 STEP 5
60 PUT (X+A,Y)-(X+A+35,Y+35),V,PSET
65 NEXT A
70 PCLS
75 GOTO 55
```

Notice that we've used the options for both GET and PUT. If you want this rocket to go faster, delete the options and switch to Mode 3.

Do-It-Yourself Program 24-1

```
5 CLS
10 FOR N = 12 TO 1 STEP -1
15 PRINT "NOTE"; N
20 PLAY STR$(N)
25 FOR I=1 TO 500: NEXT I
30 NEXT N
```

Do-It-Yourself Program 24-2

Change the following lines:

```
100 A$ = "T5;C;E;F;L1;G;P4;L4;C;E;F;L1; G"
105 B$ = "P4;L4;C;E;F;L2;G;E;C;E;L1;D"
110 C$ = "P4;L4;D+;L8;E;G;E;P8;L4;C;L8; D;
         D+"
115 D$ = "L4;E;C;L2;D3;C;L8;D3;D;L8;D2; B-"
120 E$ = "G;E;L4;G;L1;F;P4;L8;G;F;E;F"
125 F$ = "L2;G;E;L4;C;L8;D;D+;E;G;L4;A;
         L1;D3; C"
130 X$ = "XA$;XB$;XC$;XD$;XE$;XF$;"
```

Add Line 140:

```
140 PLAY X$
```

Do-It-Yourself Program 25-1

```
5   CLS: PRINT "POSITION TAPE - PRESS PLAY
    AND RECORD"
7   INPUT "PRESS <ENTER> WHEN READY"; R$
10   OPEN "O", #-1, "CHECKS"
15   CLS: PRINT "INPUT CHECKS - PRESS <XX>
     WHEN FINISHED"
20   INPUT "NUMBER :"; N$
25   IF N$ = "XX" THEN 90
30   INPUT "DATE :"; D$
40   INPUT "PAYABLE TO :"; P$
50   INPUT "ACCOUNT :"; S$
60   INPUT "AMOUNT :$"; A
70   PRINT #-1, N$, D$, P$, S$, A
80   GOTO 15
90   CLOSE #-1
92   CLS: T = 0
95   INPUT "WHICH ACCOUNT"; B$
100   PRINT "REWIND TAPE - PRESS PLAY"
110   INPUT "PRESS <ENTER> WHEN READY"; R$
120   OPEN "I", #-1, "CHECKS"
130   IF EOF(-1) THEN 170
140   INPUT #-1, N$, D$, P$, S$, A
150   IF B$ = S$ THEN T = T + A
160   GOTO 130
170   CLOSE #-1
180   PRINT "TOTAL SPENT ON -" B$, "IS $" T
```

Do-It-Yourself Program 26-1

```
10   DATA 33, 12, 42, 13, 15, 23
20   DATA 25, 30, 33, 27, 14, 8
30   DIM I(12)
40   FOR X = 1 TO 12
50   READ I(X)
60   NEXT X
70   INPUT "ITEM NO."; N
75   IF N > 12 THEN 70
80   PRINT "INVENTORY FOR ITEM" N "IS" I(N)
90   GOTO 70
```

Do-It-Yourself Program 26-2

```
5   DIM T(52)
7   DIM D(52)
10   FOR X = 1 TO 52
20   T(X) = X
30   NEXT X
34   CLS
36   PRINT @ 101, "... DEALING THE CARDS"
40   FOR X = 1 TO 52
50   C = RND(52)
60   IF T(C) = 0 THEN 50
```

```
70  O(X) = C
75  SOUND 128, 1
80  T(C) = 0
100  NEXT X
110  CLS
120  PRINT @ 107, "YOUR HAND"
130  PRINT @ 167, " "
140  FOR X = 1 TO 5
150  PRINT O(X);
160  NEXT X
```

Do-It-Yourself Program 27-1

Lines that change items:

```
110  INPUT "WHICH ITEM NO. DO YOU WANT TO
     CHANGE"; N
115  IF N > 12 THEN 110
120  INPUT "WHAT IS THE REPLACEMENT ITEM";
     S$(N)
130  GOTO 80
```

The appendix has a sample program that adds and deletes items from this list.

Do-It Yourself Program 27-2

Lines that change the song lyrics:

```
110  PRINT
120  INPUT "WHICH LINE DO YOU WANT TO
     REVISE"; L
125  IF L > 4 THEN 120
130  PRINT "TYPE THE REPLACEMENT LINE"
140  INPUT A$(L)
150  GOTO 50
```

Do-It-Yourself Program 27-3

```
1 CLEAR 1000
5 DIM A$(50)
7 CLS
10 PRINT "TYPE A PARAGRAPH"
16 :
20 PRINT "PRESS </> WHEN FINISHED"
30 X = 1
40 A$ = INKEY$
50 IF A$ = "" THEN 40
60 PRINT A$;
70 IF A$ = "/" THEN 105
80 A$(X) = A$(X) + A$
90 IF A$ = "." OR A$ = "?" OR A$ = "!" THEN X
   = X + 1
100 GOTO 40
```

```
105 PRINT: PRINT
110 INPUT "(1) PRINT OR (2) REVISE"; R
120 CLS
130 ON R GOSUB 1000, 2000
140 GOTO 105
1000 REM   PRINT PARAGRAPH
1010 FOR Y = 1 TO X-1
1020 PRINT A$(Y);
1030 NEXT Y
1040 RETURN
2000 REM   REVISE PARAGRAPH
2010 FOR Y = 1 TO X-1
2020 PRINT Y "--" A$(Y)
2030 NEXT Y
2040 INPUT "SENTENCE NUMBER TO REVISE"; S
2045 IF S > X-1 OR S < 1 THEN 2040
2050 PRINT A$(S)
2060 PRINT "TYPE PHRASE TO DELETE"
2070 INPUT D$
2080 L = LEN(D$)
2090 PRINT "TYPE A REPLACEMENT PHRASE"
2100 INPUT R$
2110 FOR Z = 1 TO LEN(A$(S))
2120 IF MID$(A$(S),Z,L) = D$ THEN 2160
2130 NEXT Z
2140 PRINT D$ "-- IS NOT IN YOUR SENTENCE"
2150 GOTO 2060
2160 E = Z - 1 + LEN(D$)
2170 A$(S) = LEFT$(A$(S),Z-1) + R$ + RIGHT
     $(A$(S),LEN(A$(S))-E)
2180 RETURN
```

Do-It-Yourself Program 27-4

Change this line to print on the printer:

```
150   PRINT #-2, A$(Y);
```

Do-It-Yourself Program 28-1

```
1 CLS: CLEAR 1000: DIM T$(100), A$(100),
  S$(100), M$(100), Z(100)
2 PRINT "POSITION TAPE -- PRESS PLAY AND
  RECORD"
4 INPUT "PRESS <ENTER> WHEN READY"; R$
8 REM
9 REM   OUTPUT TO TAPE
10 OPEN "O", #-1, "BOOKS"
15 CLS: PRINT "INPUT YOUR BOOKS -- TYPE <XX>
   WHEN FINISHED"
20 INPUT "TITLE"; T$
25 IF T$ = "XX" THEN 50
26 INPUT "AUTHOR"; A$
```

```
28 INPUT "SUBJECT"; S$
30 PRINT #-1, T$, A$, S$
40 GOTO 15
50 CLOSE #-1
60 CLS: PRINT "REWIND THE RECORDER AND PRESS
   PLAY"
70 INPUT "PRESS <ENTER> WHEN READY"; R$
74 REM
76 REM     INPUT FROM TAPE
78 B = 1
80 OPEN "I", #-1, "BOOKS"
85 IF EOF(-1) THEN 120
90 INPUT #-1, T$(B), A$(B), S$(B)
95 B = B + 1
110 GOTO 85
120 CLOSE #-1
490 PRINT
500 INPUT "SORT BY (1) TITLE (2) AUTHOR OR
    (3) SUBJECT"; A
510 IF A > 3 OR A < 1 THEN 500
520 ON A GOSUB 1000, 2000, 3000
530 GOSUB 4000
540 PRINT
550 FOR X = 1 TO B-1
560 PRINT "TITLE :" T$(Z(X))
570 PRINT "AUTHOR: " A$(Z(X))
580 PRINT "SUBJECT :" S$(Z(X))
590 NEXT X
600 PRINT : GOTO 500
800 REM
900 REM     BUILD M$ ARRAY
1000 FOR X = 1 TO B-1
1010 M$(X) = T$(X)
1020 NEXT X
1030 RETURN
2000 FOR X = 1 TO B-1
2010 M$(X) = A$(X)
2020 NEXT X
2030 RETURN
3000 FOR X = 1 TO B-1
3010 M$(X) = S$(X)
3020 NEXT X
3030 RETURN
3900 REM
4000 REM     SORT ROUTINE
4005 T = 1
4010 X = 0
4020 X = X + 1
4030 IF X > B-1 THEN RETURN
4040 IF M$(X) = "ZZ" THEN 4020
4050 FOR Y = 1 TO B-1
4060 IF M$(Y) < M$(X) THEN X = Y
4065 Z(T) = X
4080 NEXT Y
4085 T = T + 1
4090 M$(X) = "ZZ"
4100 GOTO 4010
```

Do-It-Yourself Program 29-1

```
10 DIM S$(4), N$(13), T(4,13)
20 DATA SPADES, HEARTS, DIAMONDS, CLUBS
30 FOR X = 1 TO 4
40 READ S$(X)
50 NEXT X
60 DATA ACE, 2, 3, 4, 5, 6, 7, 8, 9, 10,
   JACK, QUEEN, KING
70 FOR X = 1 TO 13
80 READ N$(X)
90 NEXT X
100 FOR S = 1 TO 4
110 FOR N = 1 TO 13
120 T(S,N) = (S-1) * 13 + N
130 NEXT N,S
140 FOR X = 1 TO 52
150 S = RND(4): N = RND(13)
160 IF T(S,N) = 0 THEN 150
170 T(S,N) = 0
180 PRINT N$(N) "-" S$(S),
190 NEXT X
```

Do-It-Yourself Program 30-1

```
5 CLS
10 FOR NUMBER = 1 TO 10
20 PRINT NUMBER ^ 2
30 NEXT NUMBER
```

Do-It-Yourself Program 30-2

```
5 CLS
10 FOR NUMBER = 100 TO 1 STEP -10
20 PRINT SQR(NUMBER)
30 NEXT NUMBER
```

Do-It-Yourself Program 30-3

```
5 CLS
10 FOR A = -180 TO 179 STEP 10
15 RD=A/57.29577951
30 CP=COS(RD)*14+16.5   'COS POSITION
40 SP=SIN(RD)*14+16.5   'SIN POSITION
50 IF SP<=CP THEN 70
60 PRINT TAB(CP); "C";TAB(SP);"S": GOTO 80
70 PRINT TAB(SP);"S";TAB(CP);"C"
80 NEXT A
90 GOTO 10
```

Do-It-Yourself Program 30-4

```
a.) ?LOG (1003)
    6.91075079
b.) ?LOG(74.9865)
    4.3173081
c.) ? LOG(3.354285)
    1.21023863
```

Do-It-Yourself Program 30-5

```
5 CLS
10 INPUT "WHAT NUMBER "; NUMBER
15 X=LOG(NUMBER)/LOG(10)
20 PRINT "THE LOG BASE 10 OF" NUMBER "IS" X
25 GOTO 10
```

a.) 7.00890077 E -11

Note: The log of 1 in any base is 0. The answer the computer displays is the result of a round-off error. All computers produce this answer.

b.) 1
c.) 2
d.) 2.69897001
e.) -1
f.) 3.00043408

Do-It-Yourself Program 30-6

```
1.) DEFFNR(X) = X*57.29577951
2.)
5 CLS
10 DEF FNC(X) = X ↑ 3
20 INPUT "WHAT NUMBER DO YOU WANT TO CUBE";X
30 X=FNC(X)
40 PRINT X
50 FOR A = 1 TO 75
55 NEXT A
60 GOTO 20
```

Do-It-Yourself Program 31-1

```
5 CLS
10 X$ = STRING$(30,"-")
20 FOR X = 64 TO 416 STEP 64
30 PRINT @ X, X$
40 PRINT @ 97, "BILL"
41 PRINT @ 161, "SUE"
42 PRINT @ 225, "JON"
43 PRINT @ 289, "MARY"
50 PRINT @ 38, "MATH"
51 PRINT @ 45, "SPELL"
```

```
52 PRINT @ 53, "READ"
60 PRINT @ 103, "X"
61 PRINT @ 175, "X"
62 PRINT @ 231, "X"
63 PRINT @ 311, "X"
70 NEXT X
80 GOTO 80
```

Do-It-Yourself Program 31-2

```
5 CLS
10 X$ = "ABCDEB"
20 Y$ = "B"
30 PRINT INSTR(X$,Y$); INSTR(4,X$,Y$)
```

Do-It-Yourself Program 31-3

```
15 X = 1
20 X$ = "JAMES SMITH,6550HARISON,DALLASTX*
   75002:SUE SIM,RT3,GRAVIOSMO*65084: LYDIA
   LONG,3445SMITHST,ASBURYNJ*32004:BOB
   STRONG,BOX 60,EDMONTONALBERTACA:TIMMY
   DUNTON, PIERMONTMO*65078"
50 P = INSTR(X,X$,A$): PRINT P
60 IF P < > 0 THEN X = P+1: GOTO 50
```

Do-It-Yourself Program 31-4

```
10 DIM TBL$(26)
20 FOR I=0 TO 25
30 READ TBL$(I):NEXT I
40 PRINT "ENTER OLD-STYLE PHONE NUMBER"
50 INPUT N$
60 IF N$=" " THEN 40
70 FOR I=1 TO LEN(N$)
80 C$=MID$(N$,I,1)
90 IF C$<"A" OR C$ >"Z" THEN 120
100 C$=TB$(ASC(C$)-65)
110 MID$(N$,I)-C$
120 NEXT I
130 PRINT "NEW-STYLE = "; N$
140 REM A B C D E F
150 DATA "2","2","2","3","3","3"
160 REM G H I J K L
170 DATA "4","4","4","5","5","5"
180 REM M N O P Q R
190 DATA "6","6","6","7","Q","7"
200 REM S T U V W X
210 DATA "7","8","8","8","9","9"
220 REM Y Z
230 DATA "9","Z"
```

Do-It-Yourself Program 32-1

```
10 A$ = "$$###,######.## DOLLARS"
```

Do-It-Yourself Program 32-2

```
5 CLS
10 INPUT "INCOME"; I
15 INPUT "EXPENSES";E
20 N = I-E    'NET GAIN OR LOSS
25 A$ = "$$#####.##"
30 B$ = "$$#####.##"
35 C$ = "+$$#####.##"
40 CLS: PRINT @ 33, "MONTHLY ECONOMIC STATUS
   REPORT"
45 PRINT @ 96, STRING$ (32,"-")
50 PRINT @ 160, "INCOME"
55 PRINT @ 256, "EXPENSES"
60 PRINT @ 352, "TOTAL (+) OR (-)"
65 PRINT @ 340, STRING$(10,"-")
70 PRINT @ 180, USING A$; I
75 PRINT @ 276, USING B$; E
80 PRINT @ 371, USING C$; N
90 GOTO 90
```

Try modifying this program to keep track of your electricity bills and to store the information on a yearly basis.

Do-It-Yourself Program 32-3

```
5 CLS
10 PRINT "THIS" TAB(POS(0)+4)"IS";
20 PRINT TAB(POS(0)+4)"EVENLY"
   TAB(POS(0)+4) "SPACED"
```

SAMPLE PROGRAMS

Sample Program #1

Type this program and save it on cassette, but don't open it (or run it) until Christmas!

```
5 CLS
10 PRINT @ 64, STRING$ (32,"*")
15 PRINT @ 352, STRING$ (32,"*")
20 PRINT @ 199, "JOY TO THE WORLD"
25 FOR X = 1 TO 1000: NEXT X
30 CLS
35 PRINT @ 64, "JOY TO THE WORLD"
40 PRINT @ 96, "THE LORD IS COME"
45 PRINT @ 128, "LET EARTH RECEIVE HER KING"
50 PRINT @ 160, "LET EVERY HEART"
55 PRINT @ 192, "PREPARE HIM ROOM"
60 PRINT @ 224, "AND HEAVEN AND NATURE SING"
65 PRINT @ 256, "AND HEAVEN AND NATURE SING"
70 PRINT @ 288, "AND HEAVEN AND HEAVEN AND
   NATURE SING"
100 A$="T4; O3; L2;C;L4;O2;B;L8;A;L2.;G;L4;
    F;L2;E;D;"
105 B$="L2.;C;P32;L4;G;L2;A;L4;P32;A;L2.;B;
    P32;L4;B;O3;L1.;C"
110 C$="L4;C;C;O2;L4;B;A;G;L4.;G;L8;F;L4;E;
    O3;C"
115 D$="O3;L4;C;O2;B;A;G;P32;L4.;G;L8;F;L4;
    E;P32;E;P32;E;P32;E;P32;E;P32;L8;E;F"
120 E$="L2.;G;L8;F;E;L4;D;P32;D;P32;D;P32;
    L8;D;E;L2.;F;L8;E;D"
125 F$="O2;L4;C;O3;L2;C;O2;L4;A;L4.;G;L8;F;
    L4; E;F;L2;E;D;L1;C"
130 X$ = "XA$;XB$;XC$;XD$;XE$;XF$;"
135 PLAY X$
200 PMODE 3,1
205 PCLS 4
210 SCREEN 1,0
215 COLOR 1,4
220 LINE (90,96)-(118,26),PSET
225 LINE (146,96)-(118,26),PSET
230 LINE (90,96)-(146,96),PSET
235 DRAW "BM112,96;D15;R10;U15"
240 LINE (0,112)-(255,96),PSET
245 PAINT (238,85),1,1
250 X = RND(255)
255 Y = RND(115)
260 A = RND(4)
265 PSET (X,Y,A): GOTO 250
```

Sample Program #2

```
1 '*** BACK TO BACH ***
2 '
```

```
5 CLS
10 PRINT @ 96, STRING$(32,"*")
20 PRINT @ 320, STRING$(32,"*")
25 PRINT @ 201, "BACK TO BACH"
40 FOR X = 1 TO 1000: NEXT X
55 A$ = "T6;O2;L2;G;L4;C;D;E;F;L2;G;C;P16;
   C"
60 B$="L2;A;L4;F;G;A;B;O3;L2;C;O2;C;P16;C;
   F;L4;G;F;E;D"
65 C$="L2;E;L4;F;E;D;C;L2;O1;B;O2;L4;C;D;
   E;C"
70 D$="L2;E;L1;D;L2;G;L4;C;D;E;F;L2;G;C;
   P16;C"
75 E$="L2;A;L4;F;G;A;B;O3;L2;C;O2;C;P16;C;
   F;L4;G;F;E;D"
80 F$="L2;E;L4;F;E;D;C;D;E;L2;F;O1;B;L1;O2;
   C"
85 X$="XA$;XB$;XC$;XD$;XE$;XF$;"
90 PLAY X$
```

Sample Program #3

```
1 ' ***MEXICAN HAT DANCE***
2 '
5 CLS
10 PRINT @ 96,STRING$(32,"*")
20 PRINT @ 320,STRING$(32,"*")
30 PRINT @ 199,"MEXICAN HAT DANCE"
40 FOR X = 1 TO 500: NEXT X
125 REM S T A R T T U N E
130 O$="V15;T3;O2;"
135 P$="L8CFP8CFP8CFP4P8"
140 Q$="CFGFEP8FGP4P8"
145 X$="XO$;XP$;XQ$;"
150 PLAY X$
155 R$="CEP8CEP8CEP4P8
160 S$="CEFEDP8EFP4P8
165 Y$="XO$;XR$;XS$;"
170 PLAY Y$
180 REM 2ND TIME
185 O$= "V25;T3;O1"
190 PLAY X$
195 O$="T3;O4"
197 S$="CEFEDP8EFO4CO3AF"
200 PLAY Y$
210 A$="O3CO2BO3CO2AA-AFEFCP4"
220 B$="CO1BO2CDEFGAB-O3CEG"
225 O$="V15;T4;"
230 Z$="XO4;XA$;XB$;"
235 PLAY Q$
240 C$="O3B-AB-GF+FEG=ECEG"
245 D$="O4L16CP16CP16CP16L8DCO3B-AGFP4"
250 E$="XO$;XC$;XD$;"
255 PLAY F$
260 F$="O2L16GP16GP16GP16DP16DP16DP16EP16FP
    16L8EL16GP16O1GP16L8G
```

```
265 G$="V15O2L16GP16GP16GP16DP16DP16DP16EP
    16FP16L8ECO1GC"
270 H$="XF$;XG$;"
280 PLAY H$
285 I$="XF$;"
290 PLAY I$
295 J$="O2L16GP16GP16GP16AP16GP16GP16AP16BP
    16O3L4CP8"
300 PLAY "XJ$;"
310 K$="O4L1DL4DEDEL8DEDEL16DEDEDEDEL
    32DEDEDEDEDEDEDEDEL64DEDEDE
    DEDEDEDEDEDEDEDEDEL32DD-CO3BB-AA-
    GF+FEE-DD-L4DD-"
320 PLAY "XK$;"
330 M$="T5L8DO2BB-BGF+GL4DP8"
340 N$="L8DC+DEF+GABO3CO2L4AP8"
350 AA$="O3L8CO2BOECO2AG+AF+FF+L4DP8"
370 BB$="O3L8DDDEDCO2BAO3DEDCO2BA"
380 CC$="O2DEDCO1BAO4DEDDEDDEDDEF+
    GDO3BGT4DO2BGT3DO1T2BL4P2V30L1G"
400 PLAY "XM$;XN$;XAA$;XBB$;XCC$;"
500 PMODE 4,1
505 FOR Y = 1 TO 5
510 SCREEN 1,0
520 PCLS
550 CIRCLE (128,96),50,1,,2,,85,,67
560 CIRCLE (128,96),25,1,2,,5,1
570 LINE (105,96)-(151,96),PSET
600 PMODE 4,1
610 SCREEN 1,0
620 PCLS
630 CIRCLE (128,75),50,1,,2,,85,,67
660 CIRCLE (128,75),1,2,,5,1
670 LINE (105,75)-(151,75),PSET
675 NEXT Y
680 IF Y > 5 THEN 690
685 GOTO 500
690 CLS
700 PRINT @ 227, "NOW THAT'S A HOT TAMALE"
710 FOR X = 1 TO 600:NEXT X
720 GOTO 5
```

Sample Program #4

```
1 ' ***BUFFALO GALS***
2 '
5 CLS
10 PRINT @ 64, STRING$(32,"*")
15 PRINT @ 384, STRING$(32,"*")
20 PRINT @ 201, "BUFFALO GALS"
25 FOR X = 1 TO 1000: NEXT X: CLS
30 PRINT @ 32, "AS I WAS WALKING DOWN THE
   STREET"
35 PRINT @ 64, "DOWN THE STREET, DOWN THE
   STREET"
40 PRINT @ 96, "A PRETTY GAL I HAPPENED"
```

```
45 PRINT @ 133, "TO MEET"
50 PRINT @ 160, "JUST AS LOVELY AS"
55 PRINT @ 197, "THE MORNING DEW"
60 PRINT @ 224, "BUFFALO GALS WON'T YOU"
65 PRINT @ 261, "COME OUT TONIGHT"
70 PRINT @ 288, "COME OUT TONIGHT,"
75 PRINT @ 320, "COME OUT TONIGHT,"
80 PRINT @ 352, "BUFFALO GALS WON'T YOU"
85 PRINT @ 391, "COME OUT TONIGHT"
90 PRINT @ 416, "AND DANCE IN THE"
95 PRINT @ 453, "LIGHT OF THE MOON." ,
100 A$="T4;C;E;P32;E;F;P32;F;A;G;L2;E;"
105 B$="L4;G;F;L2;D;L4;A;G;E;C;"
110 C$="L4;E;P32;E;F;P32;F;L8;A;P32;A;L4;
    G;E;O3;L8;C;P32;C;"
115 D$="O2;B;P32;B;G;P32;G;L4;F;O1;B;O2;
    L1;C;P16;"
120 E$="L8;C;P32;C;L4;P32;C;E;L8;G;P32;G;
    A;P32;A;L4;G;L2;E"
125 F$="L8;G;P32;G;L4;F;L2;D;L4;A;L8;G;
    P32;G;L2;E"
130 G$="L8;C;P64;C;P64;L4;C;E;L8;G;P32;G;
    L4;A;L8;G;P32;G;L4;E;O3;C;"
135 H$="O2;B;L8;G;P32;G;F;P32;F;L4;D;L2.;
    C;"
140 X$ = "XA$;XB$;XC$;XD$;XE$;XF$;XG$; XH$;"
145 PLAY X$
150 CLS
155 PRINT @ 230, "THAT'S ALL FOLKS"
```

Sample Program #5

```
1 '*** IN-OUT ***
2 '
5 PMODE 3,1
10 PCLS3
15 SCREEN 1,0
20 FOR I = 3 TO 7
25 FOR J = 2 TO 6
30 FOR S = 0 TO 3
35 FOR R = 0 TO 3
40 COLOR R,S
45 A = 0:B=255:C=0:D=191
50 LINE (A,C)-(B,D),PSET,B
55 A=A+J:B=B-J:C=C+I:D=D-I
60 IF A<255 AND C<191 THEN 50
65 NEXT R
70 NEXT S
75 NEXT J,I
80 GOTO 30
```

Sample Program #6

```
1 '*** DRAWING TRIANGLES ***
10 CLS: CLEAR
75 PRINT @96,STRING$(32,"*")
```

```
80 PRINT @ 288, STRING$(32,"*")
100 PRINT @ 160, "THIS PROGRAM DRAWS THE
    TRIANGLE YOU SPECIFY AND THEN CALCULATES
    ITS AREA"
110 FOR X=1 TO 2200: NEXT: CLS
120 CLS:PRINT"FOR 3 SIDES TYPE, SSS (0-100)"
125 PRINT"FOR 2 SIDES (1-100) AND 1 ANGLE (0-
    90) TYPE, SAS"
130 PRINT "FOR 1 SIDE (0-60) AND 2 ANGLES (0-
    90) TYPE, ASA"
140 INPUT A$: IF A$="SAS" GOTO 300
150 IF A$="ASA" GOTO 400
200 'SSS
210 PRINT "ENTER 3 SIDES, (LONGEST SIDE
    FIRST)"
220 INPUT L1,L2,L3
225 IF L2>L1 OR L3>L1 THEN PRINT "***LONGEST
    FIRST PLEASE . . .": PRINT: GOTO 210
230 S=(L1+L2+L3)/2
235 IF S<=L1 THEN PRINT "***NOT A
    TRIANGLE***": PRINT: GOTO 210
240 Y3=2*SQR(S*(S-L2)*(S-L1)*(S-L3))/L1
250 A=Y3/L2: A=ATN(A/SQR(-A*A+1))
260 X3=COS(A)*L2
270 AR=(L1*Y3)/2
280 GOTO 490
300 'SAS
310 PRINT "ENTER 2 SIDES AND 1 ANGLE: AB,AC,
    THETA (LARGEST SIDE FIRST)"
320 INPUT L1,L2,T
325 T=(T*3.14159)/180
330 Y3=L2*SIN(T)
340 X3=COS(T)*L2
350 AR=(L1*Y3)/2: GOTO 490
400 'ASA
410 PRINT "ENTER 2 ANGLES AND 1 SIDE: THETA1,
    THETA2, AB"
420 INPUT T1,T2,L2
425 T1=(T1*3.14159)/180: T2=(T2*3.14159)/
    180
430 Y3=L2*SIN(T1)
440 B1=COS(T1)*L2
450 B2=Y3/TAN(T2)
460 L1=B1+B2: X3=B1: IF L2>L1 THEN X=L1:
    L1=L2: L2=X
470 AR=(L2*Y3)/2
490 CLS:PMODE4,1:PCLS:SCREEN 1,1
500 F=1
510 VC=(3.14159 * (L1*F-X3*F)*(Y3*F)^2)/3
520 VS=(3.14159 *(X3*F)*(Y3*F)^2)/3:
    VT=VC+VS
530 S1=Y3/X3: S2=Y3/(X3-L1)
532 IF INT(X3) = 0 THEN 1100
533 IF INT(X3)=INT(L1) THEN 1000
535 IF X3>L1 THEN 1199
537 IF X3=L2 THEN 1000
540 FOR Y=20 TO L1*2+20 STEP 2:
```

```
    PSET(Y,Y3+5,5): NEXT
550 FOR X=0 TO X3
551 PSET(X*2+20,S1*(X3-X)+5,5): NEXT
560 FOR X=X3 TO L1: PSET(X*2+20,Y3+(S2* (L1-
    X)+5),5): NEXT
580 FOR X=1 TO 600: NEXT X
610 PRINT @ 130,"AREA=";AR;" SQ. UNITS";
630 PRINT @352, "*";: INPUT "TO RUN AGAIN,
    PRESS <1> <ENTER>"; B6: IF B6=1 THEN 120
640 STOP: GOTO 10
1000 FOR Y=5 TO Y3+5: PSET(X3*2+20,Y,4):
     NEXT: GOTO 540
1100 FOR Y=5 TO Y3+5: PSET(20,Y,5): NEXT:
     GOTO 540
1200 FOR X=L1 TO X3: PSET(X*2+20,Y3+(S2*(L1-
     X)+5),5): NEXT: GOTO 540
1300 FOR X=X3 TO 0: PSET(X*2+20,Y3+(S1*(0-
     X)+5),4): NEXT: GOTO 540
```

Sample Program #7

```
1 '*** PROJECTION STUDIES ***
2 '
5 PMODE 4,1
10 PCLS
15 SCREEN 1,0
20 DRAW "BM50,50R60D10NL20D20L20NU20L20NU
   20L20U20NR20U10" 'TOP VIEW
25 DRAW"BM50,100R20ND20R20ND20R20D20NL20D
   10L60U10NR20U20" 'FRONT VIEW
30 DRAW "BM150,100R30D30L30U10NE20U20"
   'SIDE VIEW
35 ' OBLIQUE VIEW_LINES 40-60
40 DRAW "BM150,50U5E15R10BF20BD30NR5L20H
   25U10
45 DRAW"BM150,50U5F8U15R15H8F8L15F8NR15D
   15F8ND10E15NR10H8
50 LINE (175,30)-(200,55),PSET
55 LINE -(200,80),PSET
60 LINE (167,60)-(183,46),PSET
65 GOTO 65
```

Sample Program #8

```
1 '*** UNFOLDING BOX ***
2 '
5 PCLEAR 8
10 PMODE 3,1
15 PCLS
20 COLOR 6,5
25 DRAW"BM100,100U30NR30E15GR30NG15D30G16
   NU30L30"
30 PAINT (105,95),8,6
35 PAINT (135,80),8,6
```

```
40 PAINT (110,65),8,6
45 SCREEN 1,1
50 FOR X = 1 TO 600: NEXT X
110 PMODE 3,5
112 PCLS
115 COLOR 6,5
120 DRAW "BM100,100U30NR30E20R30G20D30NL
    30F20L30H20
125 LINE (100,100)-(70,95),PSET
130 LINE -(70,65),PSET
135 LINE -(100,70),PSET
140 LINE (70,95)-(40,65),PSET,B
145 LINE (130,100)-(160,95),PSET
150 LINE -(160,65),PSET
155 LINE -(130,70),PSET
160 PAINT (95,95),8,6
165 PAINT (105,95),8,6
170 PAINT (135,85),8,6
175 PAINT (45,85),8,6
180 PAINT (115,65),8,6
185 PAINT (125,114),8,6
190 SCREEN 1,1
195 FOR X = 1 TO 600: NEXT X
200 GOTO 10
```

Sample Program #9

```
1 '*** SINE WAVE ***
2 '
5 PMODE 4,1
10 PCLS
15 SCREEN 1,1
20 LINE (0,86)-(255,86),PSET
25 PI=3.14159
30 A1=-4*PI
35 A2=4*PI
40 N=180
45 R=50
50 X=(A2-A1)/N
55 F=255/(A2-A1)
60 FOR I =A1 TO A2 STEP X
65 X=I*F
70 Y=R*SIN(I)
75 PSET ((X+140),(80+Y),1)
80 NEXT I
90 GOTO 90
```

Sample Program #10

```
1 '*** SIN/COS ***
2 '
10 PMODE 4,1
20 PCLS
```

```
30 SCREEN 1,0
40 LINE (127,5)-(127,185),PSET
50 LINE (7,95)-(247,95),PSET
60 FOR XSCALE=7 TO 247 STEP 20
70 PRESET (XSCALE,95)
80 NEXT XSCALE
90 FOR YSCALE=5 TO 185 STEP 10
100 PRESET(127,YSCALE)
110 NEXT YSCALE
130 FOR X=-180 TO 180 STEP 1.5
140 AX=X/57.29578
145 XP=X/1.5+127
150 F1=-(SIN(AX)*90)+95
160 F2=-(COS(AX)*90)+95
170 PSET(XP,F1,1): PSET(XP,F2,1)
180 NEXT X
190 GOTO 190
```

Sample Program #11

```
1 '*** RANDOM GRAPHICS ***
2 '
10 PMODE 3,1
15 PCLS
20 SCREEN 1,1
25 F =RND(4):B=RND(8): IF B=F OR (B-4=F)
   THEN 25
30 COLOR F,B:PCLS B: FOR L = 0 TO 5
35 LINE -(RND(255),RND(191)),PSET
40 CIRCLE (RND(255),RND(191)),RND(100)
50 NEXT: FOR P=0 TO 10
55 PAINT (RND(255),RND(191)),RND (4),F
60 NEXT: FOR H = 1 TO 7
65 FOR T=0 TO 600: NEXT T: GOTO 10
```

Sample Program #12

```
1 '***NAVAHO BLANKET***
2 '
5 PMODE 3,1
10 PCLS 4
15 SCREEN 1,0
20 COLOR 1,0
25 FOR X = 0 TO 255 STEP 18
30 OY = Y
35 Y = 30-OY
40 LINE (X,100-Y)-(X+10,100-OY),PSET
45 LINE(X,120+Y)-(X+10,120+OY),PSET
50 NEXT
60 FOR C = 2 TO 8
65 PAINT (0,110),C,1
70 NEXT
80 GOTO 5
```

Sample Program #13

```
1 '*** PAINTED LACE ***
2 '
5 PMODE 3,1
10 PCLS
20 SCREEN 1,1
30 DRAW"BM50,180U60BU20U60R60BR20R60D60
   BD20D60L60BL20L60
40 DRAW"BM50,180U60R40BR20R80D20BL20L60
   BL20L20D20R20BR60R20U20
50 DRAW"BM50,180R60U80BU20U40L40BD20D20
   BD60D20R20U60BU20U20L20
60 DRAW"BM50,180U60BU40BR20R60BR20R20U20
   L20D60BD20D20R20
70 DRAW"BM50,180BR80U40BU20U80
80 DRAW"BM50,180BU80R80BR20R40
90 PAINT (85,128),6,8
95 PAINT (95,78),6,8
97 PAINT (155,95),6,8
98 PAINT (135,145),6,8
99 PAINT (128,185),7,8
100 PAINT (75,150),7,8
101 PAINT (160,150),7,8
102 PAINT (75,75),7,8
103 PAINT (160,75),7,8
104 PAINT (120,110),7,8
110 FOR X =1 TO 600: NEXT X
200 GOTO 5
```

Sample Program #14

```
1 '*** DRAWING BOARD ***
2 '
3 CLS
5 PRINT @128,STRING$(32,"*"):PRINT@ 288,
  STRING$(32,"*")
10 PRINT @ 200, "DRAWING BOARD"
15 FOR X = 1 TO 600: NEXT X
20 CLS
25 PRINT @ 96, "PRESS <↑> FOR UP, <DOWN
   ARROW> FOR DOWN, <BACKSPACE> FOR LEFT,
   <TAB> FOR RIGHT, <A> FOR SOUTHWEST, <S>
   FOR SOUTHEAST, <W> FOR NORTHEAST, <Q> FOR
   NORTHWEST"
30 PRINT @ 288,"PRESS <1> FOR INVISIBLE
   LINE, <2>,<3>, OR <4> FOR DIFFERENT
   COLORED VISIBLE LINES, PRESS </> TO
   CHANGE COLOR-SET"
35 PRINT @ 448, "PRESS <SPACEBAR> TO PAUSE"
40 FOR X=1 TO 4800: NEXT X
45 CC=4: TG=0
50 PMODE 3,1
55 PCLS
```

```
60 SCREEN 1,TG
70 X=128:Y=96:XI=0:YI=0
80 U$="^": D$=CHR$(10): W$=CHR$(8):
   E$=CHR$(9)
90 NW$="Q": NE$="W": SW$="A": SE$="S"
100 C1$="1":C2$="2":C3$="3": C4$="4"
110 A$=INKEY$
120 IF A$=U$ THEN YI=-1:XI=0: GOTO 240
130 IF A$=D$ THEN YI=1:XI=0: GOTO 240
140 IF A$=W$ THEN XI=-1:YI=0: GOTO 240
150 IF A$=E$ THEN XI=1:YI=0: GOTO 240
160 IF A$=NE$ THEN XI=1:YI=-1: GOTO 240
170 IF A$=NW$ THEN XI=-1:YI=-1:GOTO 240
180 IF A$=SE$ THEN XI=1:YI=1:GOTO 240
190 IF A$=SW$ THEN XI=-1:YI=1:GOTO 240
200 IF C1$<=A$ AND A$<=C4$ THEN CC=ASC(A$)-
    48: GOTO 240
210 IF A$="/" THEN TG=(NOT TG AND 1) OR (TG
    AND NOT 1): GOTO 240
220 SCREEN 1, TG
230 IF A$=" " THEN XI=0: YI=0
240 X=X+XI:Y=Y+YI:IF X<0 THEN X=0
250 IF X>255 THEN X=255
260 IF Y<0 THEN Y=0
270 IF Y>191 THEN Y =191
275 IF CC=1 THEN PSET(X,Y,3)
280 PSET (X,Y,CC)
290 GOTO 110
```

Sample Program #15

```
1 '*** INTERACTING LINES ***
2 '
5 CLS
20 C = C + 1
25 IF C > 8 THEN C = 5
30 COLOR C,1
50 PRINT "TYPE X0,Y0";
60 INPUT X0,Y0
70 PRINT "TYPE X1,Y1";
80 INPUT X1,Y1
90 PMODE 3,1
95 PCLS
100 SCREEN 1,1
110 LINE (X0,Y0)-(X1,Y1),PSET
115 FOR X = 1 TO 2000: NEXT X
120 GOTO 20
```

Sample Program #16

```
1 '*** RANDOM LINES ***
2 '
20 PMODE 4,1
25 PCLS
```

```
30 SCREEN 1,1
35 X = RND(255): Y = RND(191)
40 LINE -(X,Y),PSET
45 FOR X = 1 TO 200: NEXT X
50 GOTO 35
```

Sample Program #17

```
1 '*** 8-LEAF CLOVER ***
2 '
5 PCLEAR 8
10 PMODE 4,1
15 PCLS
20 SCREEN 1,0
25 PI=3.14159
30 A1=0: A2=2*PI
35 N=360:A=50
40 X = (A2-A1)/N
45 FOR I = A1 TO A2 STEP X
50 R = A * COS (4*I)
55 X =R *SIN(I)
60 Y =R * COS(I)
65 PSET(128 + X,96+Y,5)
70 NEXT I
75 GOTO 25
```

Sample Program #18

```
1 '*** TIMEBOMB ***
2 '
10 PMODE 4, 1
15 PCLS
20 SCREEN 1,1
25 CIRCLE (128,96),80
30 CIRCLE (128,96),90
35 PAINT (0,0),5
40 FOR T=30 TO -30 STEP -1
45 A=(2*3.1415)*T/60
50 LINE (128,96)-(75*SIN(A)+128,75*
   COS(A)+96),PSET
55 SOUND Q*2+1,20/(Q+1)+1
60 LINE (128,96)-(75*SIN(A)+128,75*
   COS(A)+96),PRESET
65 Q=60-2*T:FOR Y=Q TO 0 STEP -1:NEXT
70 NEXT
75 CLS
80 PCLS
85 PRINT @ 237,"BOOM!"
90 SOUND 1,30
95 PMODE 4,1
100 SCREEN 1,1
105 FOR I =2 TO 200 STEP 2
110 CIRCLE (128,96),I
115 NEXT I
```

```
120 SCREEN 1,1
125 FOR X =2 TO 200 STEP 2
130 CIRCLE (128,96),X,,3
135 NEXT X
140 FOR I = 2 TO 200 STEP 2
145 CIRCLE (128,96),I,3,,5
150 NEXT I
155 GOTO 155
```

Sample Program #19

```
1 '*** ROTATING FAN ***
2 '
5 PCLEAR 8
50 GOTO 600
60 LINE ((255-X),(191-Y))-(X,Y),PSET
61 J = J+1:IF J>A THEN J=0:A=RND(50)
63 RETURN
600 REM ROTATING FAN
601 FOR I = 1 TO 5 STEP 4
602 PMODE 3,I
603 PCLS
604 SCREEN 1,0
605 A=25:X=0: Y=0: J=0
610 FOR X =0 TO 254
612 COLOR X/32+1,5
615 GOSUB 60: NEXT X
620 FOR Y =0 TO 190
623 COLOR Y/24+1,5
625 GOSUB 60: NEXT Y
630 FOR X = 255 TO 1 STEP -1
633 COLOR X/32+1,5
635 GOSUB 60: NEXT X
640 FOR Y = 191 TO 1 STEP -1
643 COLOR Y/24+1,5
645 GOSUB 60: NEXT Y
650 NEXT I
660 FOR I = 1 TO 5 STEP 4
670 PMODE 3,I
680 SCREEN 1,0
690 FOR T = 1 TO 30: NEXT T
700 NEXT I
710 GOTO 660
```

Sample Program #20

```
1 '***WALKING TRIANGLES ***
10 FOR A = 90 TO 0 STEP -4
15 S1=A*9: S2=191
20 A3=A/57.29578
30 X1=0:Y1=191
40 X2=S1+X1: Y2=Y1
50 X3=X1+S2*COS(A3):Y3=Y1-S2*SIN(A3)
55 GOSUB 1000
90 NEXT A
```

```
99 GOTO 99
1000 PMODE 4,1
1005 PCLS
1010 SCREEN 1,0
1020 LINE (X1,Y1)-(X2,Y2),PSET
1030 LINE -(X3,Y3),PSET
1040 LINE -(X1,Y1),PSET
1060 RETURN
```

Sample Program #21

```
1 '*** COUNTING ***
2 '
10 CLS
20 CLEAR 1000
30 PRINT "WHERE DO YOU WANT TO START
   COUNTING?"
35 INPUT A$
40 P=LEN(A$)
50 PRINT:PRINT A$
60 C=VAL(MID$(A$,P,1))+1
70 MS$=A$: MR$=RIGHT$(STR$(C),1): PS=P:
   GOSUB 200: A$=MS$
80 IF C<10 THEN 40
90 P=P-1
100 IF P=0 THEN IF LEN(A$)=255 THEN PRINT
    "OVERFLOW": END: ELSE A$="1"+A$: GOTO 40
110 GOTO 60
200 LS=LEN(MS$)
210 IF LS< > LEN(MR$)+LS-1 OR PS<1 THEN STOP
220 MS$=LEFT$(MS$,PS-1)+MR$+RIGHT$ (MS$,LS-
    PS)
230 RETURN
```

Inventory Shopping List

```
5 CLEAR 2000: DIM S$(100)
10 REM      INVENTORY/SHOPPING LIST
20 CLS
30 PRINT @ 71, "DO YOU WANT TO--"
40 PRINT @ 134, "(1) INPUT ITEMS"
50 PRINT @ 166, "(2) REPLACE ITEMS"
60 PRINT @ 198, "(3) ADD TO THE LIST"
70 PRINT @ 230, "(4) DELETE ITEMS"
80 PRINT @ 262, "(5) PRINT ALL ITEMS"
90 PRINT @ 294, "(6) SAVE ITEMS ON TAPE"
100 PRINT @ 326, "(7) LOAD ITEMS FROM TAPE"
110 PRINT @ 395, "(1-7)";
120 INPUT M
130 IF M < 0 OR M > 7 THEN 10
140 ON M GOSUB 1000, 2000, 1020, 3000, 4000,
    5000, 6000
150 GOTO 10
900 REM
1000 REM      INPUT/ADD ITEMS
```

```
1010 Y = 1
1020 CLS: PRINT @ 8, "INPUT/ADD ITEMS"
1030 PRINT @ 34, "PRESS <ENTER> WHEN
     FINISHED"
1040 PRINT: PRINT "ITEM" Y;
1045 INPUT S$(Y)
1050 IF S$(Y) = " " THEN RETURN
1060 Y = Y + 1
1070 GOTO 1040
1900 REM
2000 REM      REPLACE ITEMS
2005 N = 0
2010 CLS: PRINT @ 9, "REPLACE ITEMS"
2020 PRINT @ 34, "PRESS <ENTER> WHEN
     FINISHED"
2030 PRINT: INPUT "ITEM NO. TO REPLACE"; N
2040 IF N = 0 THEN RETURN
2050 INPUT "REPLACEMENT ITEM"; S$(N)
2060 GOTO 2000
2900 REM
3000 REM      DELETE ITEMS
3005 N = 0
3010 CLS: PRINT @ 9, "DELETE ITEMS"
3020 PRINT @ 34, "PRESS <ENTER> WHEN
     FINISHED"
3030 PRINT: INPUT "ITEM TO DELETE"; N
3035 IF N > Y-1 THEN 3030
3040 IF N = 0 THEN RETURN
3050 FOR X = N TO Y-2
3060 S$(X) = S$(X+1)
3070 NEXT X
3080 S$(X) = " "
3090 Y = Y-1
3100 GOTO 3000
3900 REM
4000 REM      PRINT ITEMS
4010 FOR X = 1 TO Y-1 STEP 15
4020 FOR Z = X TO X+14
4030 PRINT Z; S$(Z)
4040 NEXT Z
4050 INPUT "PRESS <ENTER> TO CONTINUE"; C$
4060 NEXT X
4070 RETURN
4900 REM
5000 REM      SAVE ITEMS ON TAPE
5010 CLS: PRINT @ 135, "SAVE ITEMS ON TAPE"
5020 PRINT @ 234, "POSITION TAPE"
5030 PRINT @ 294, "PRESS PLAY AND RECORD"
5040 PRINT @ 388, "PRESS <ENTER> WHEN READY"
5050 INPUT R$
5060 OPEN "O", #-1, "LIST"
5070 FOR X = 1 TO Y-1
5080 PRINT #-1, S$(X)
5090 NEXT X
5100 CLOSE #-1: RETURN
5900 REM
6000 REM      LOAD ITEMS FROM TAPE
```

Speed Reading

```
10 REM      SPEED READING
20 CLS: PRINT @ 32, "HOW MANY WORDS PER
   MINUTE"
30 INPUT "DO YOU READ"; WPM
40 FOR X = 1 TO 23
60 READ A$ : PRINT @ 256, A$
70 FOR Y = 1 TO (360/WPM) * 460 : NEXT Y
80 REM      Y LOOP SETS LINES/MIN
90 NEXT X : END
100 DATA SCARLETT OHARA WAS NOT BEAUTIFUL
110 DATA BUT MEN SELDOM REALIZED IT WHEN
120 DATA CAUGHT BY HER OWN CHARM AS THE
130 DATA TARLETON TWINS WERE, IN HER FACE
140 DATA WERE TOO SHARPLY BLENDED
150 DATA THE DELICATE FEATURES OF HER
160 DATA "MOTHER, A COAST ARISTOCRAT OF"
170 DATA "FRENCH DESCENT, AND THE HEAVY"
180 DATA ONES OF HER FLORID IRISH FATHER
190 DATA "BUT IT WAS AN ARRESTING FACE,"
200 DATA "POINTED OF CHIN, SQUARE OF JAW"
210 DATA HER EYES WERE PALE GREEN
220 DATA "WITHOUT A TOUCH OF HAZEL,"
230 DATA STARRED WITH BRISTLY BLACK
240 DATA LASHES AND SLIGHTLY TILTED
250 DATA "THE ENDS, ABOVE THEM, HER THICK"
260 DATA "BLACK BROWS SLANTED UPWARDS,"
270 DATA CUTTING A STARTLING OBLIQUE LINE
280 DATA IN HER MAGNOLIA-WHITE SKIN--THAT
290 DATA "SKIN SO PRIZED BY SOUTHERN WOMEN"
300 DATA AND SO CAREFULLY GUARDED WITH
310 DATA "BONNETS, VEILS, AND MITTENS"
320 DATA AGAINST HOT GEORGIA SUNS
```

Memory Test

This program uses an array to test both yours and your computer's memory:

```
5 DIM A(7)
10 PRINT "MEMORIZE THESE NUMBERS"
15 PRINT "YOU HAVE 10 SECONDS"
20 FOR X = 1 TO 7
30 A(X) = RND(100)
40 PRINT A(X)
50 NEXT X
60 FOR X = 1 TO 460 * 10 : NEXT X
70 CLS
80 FOR X = 1 TO 7
90 PRINT "WHAT WAS NUMBER" X
100 INPUT R
110 IF A(X) = R THEN PRINT "CORRECT" ELSE
    PRINT "WRONG - IT WAS" A(X)
120 NEXT X
```

ASCII Character Codes

These are the ASCII codes for each of the characters on your keyboard. The first column is the character; the second is the code in decimal notation; and the third converts the code to a hexadecimal (16-based number).

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACEBAR	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D
N	78	4E

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
↑*	94	5E
↓*	10	OA
←*	8	O8
→*	9	O9
BREAK	03	03
CLEAR	12	0C
ENTER	13	0D

*If shifted, the codes for these characters are as follows: CLEAR is 92 (hex 5C); ↑ is 95 (hex 5F); ↓ is 91 (hex 5B); ← is 21 (hex 15); and → is 93 (hex 5D).

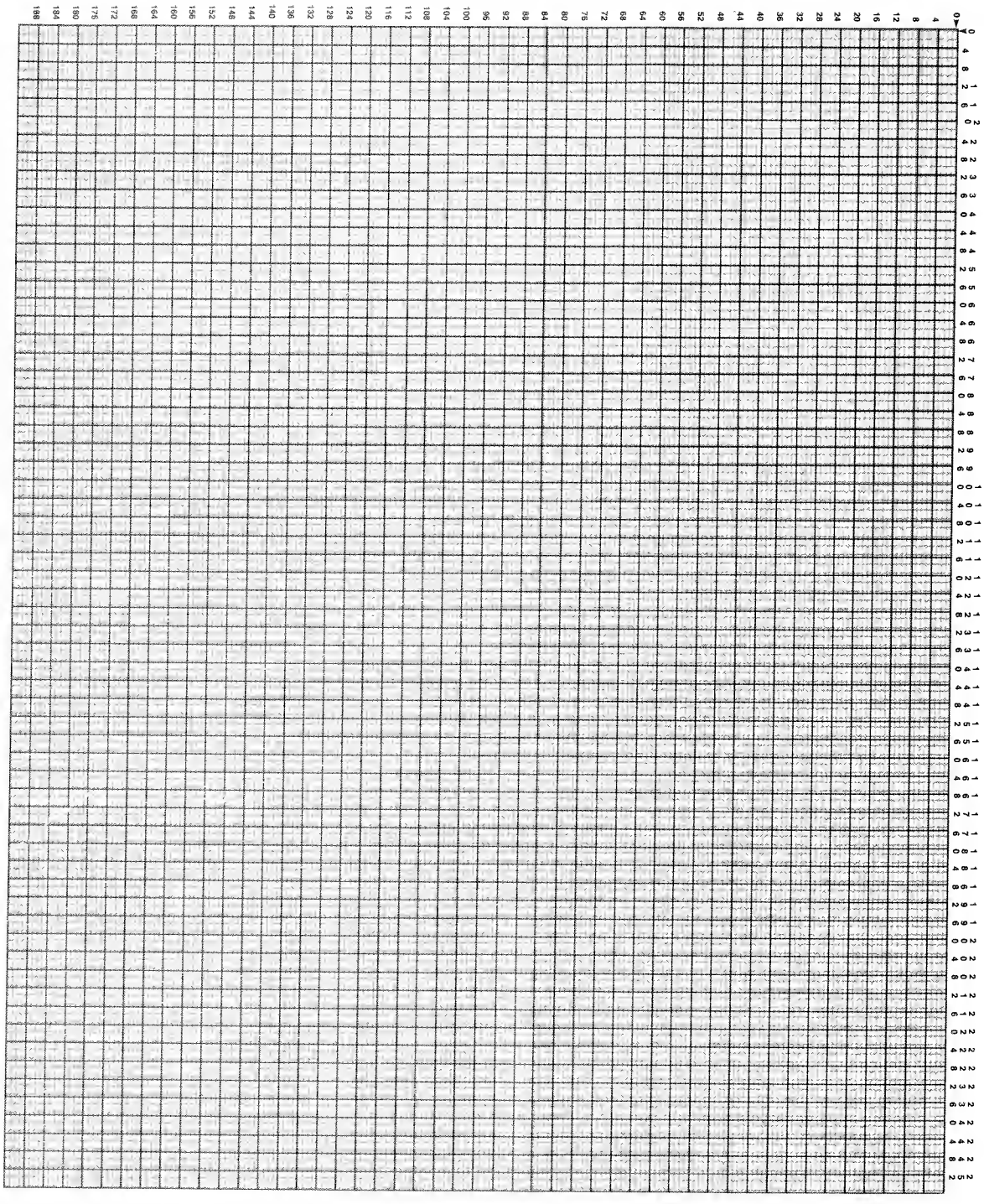
Lowercase Codes

These are the ASCII codes for lowercase letters. You can produce these characters by pressing the SHIFT and 0 keys simultaneously to get into an upper- lowercase mode. The lowercase letters will appear on your screen in reversed colors (green with a black background).

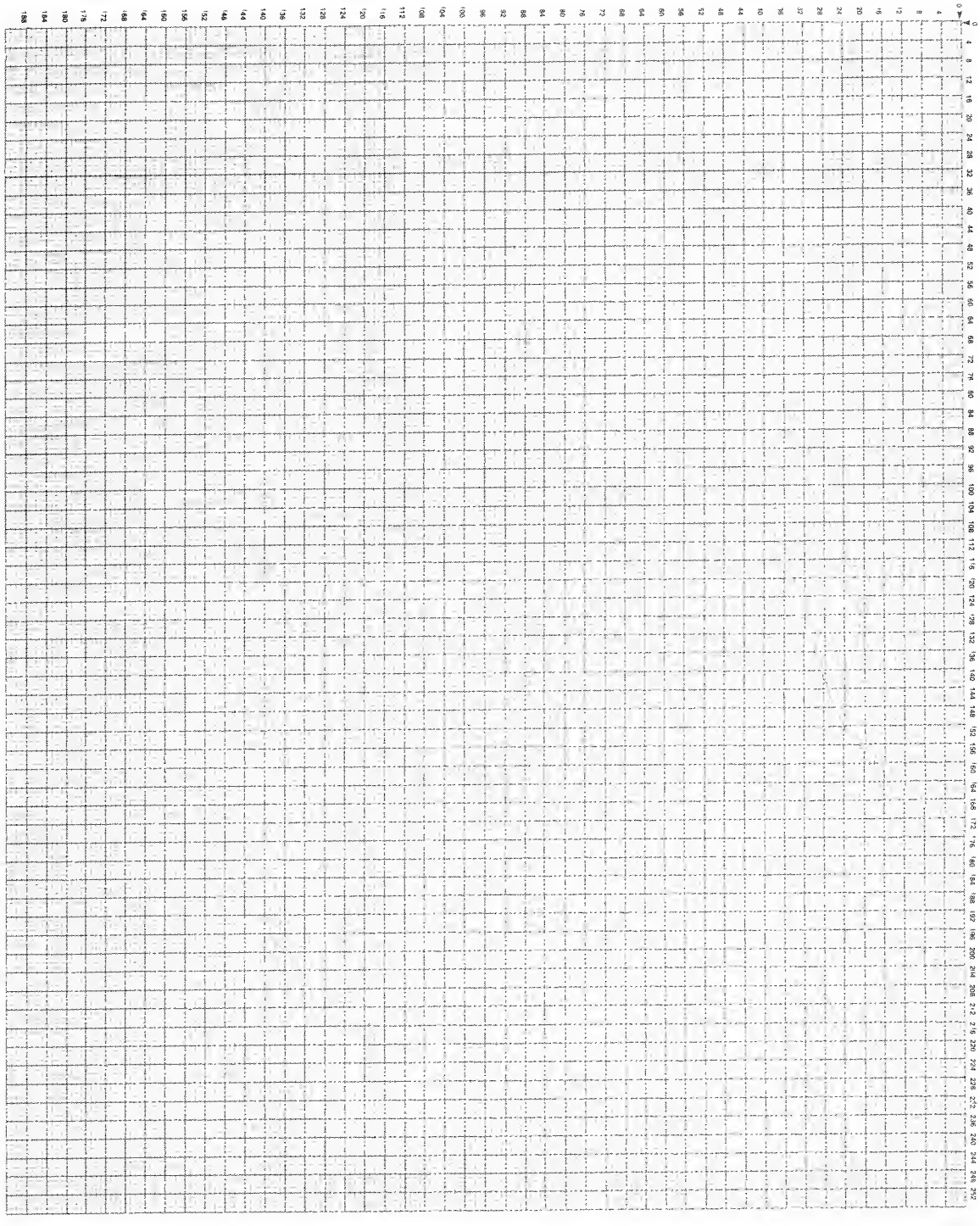
CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A

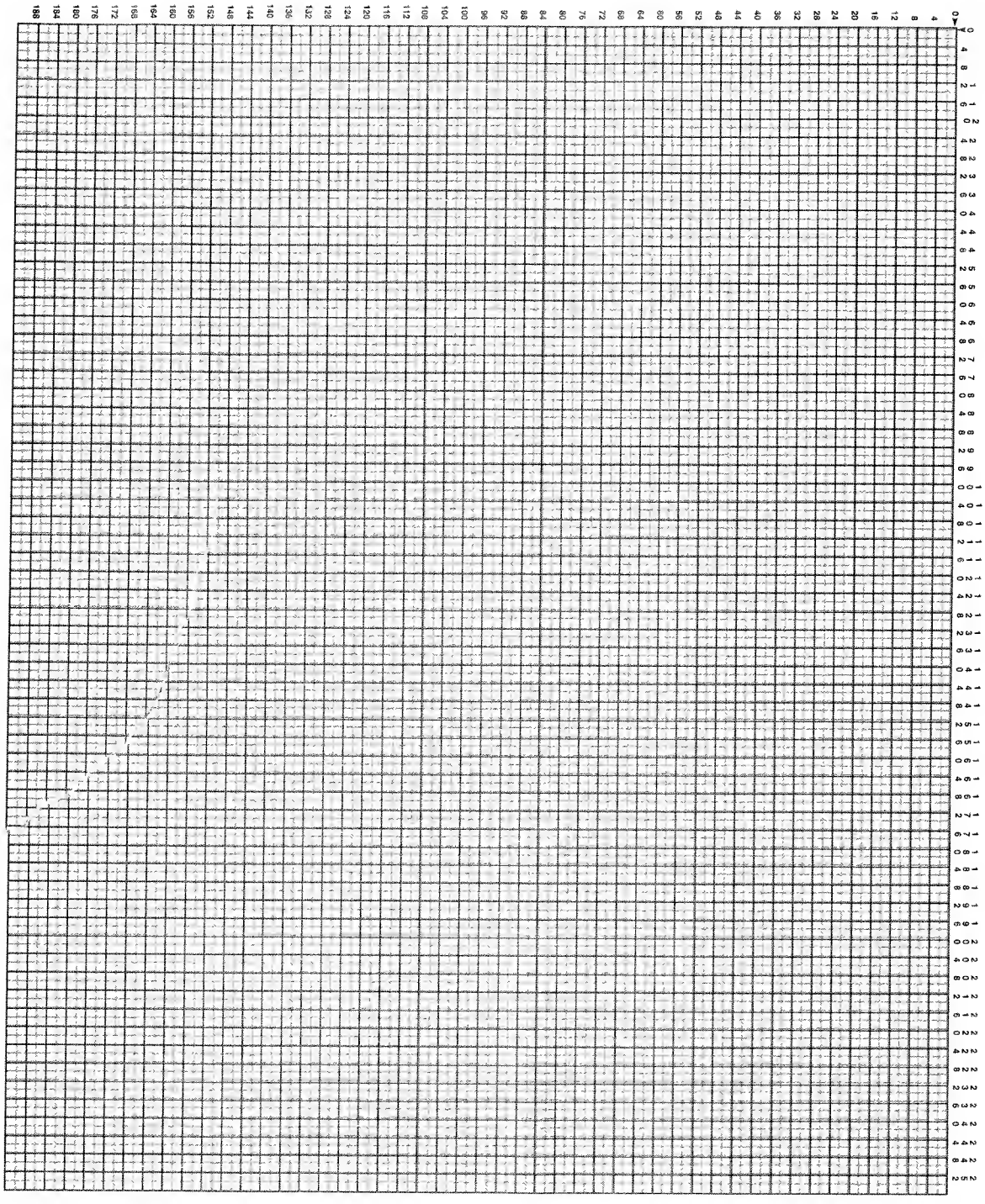
GRAPHICS SCREEN WORKSHEET (128 x 192)



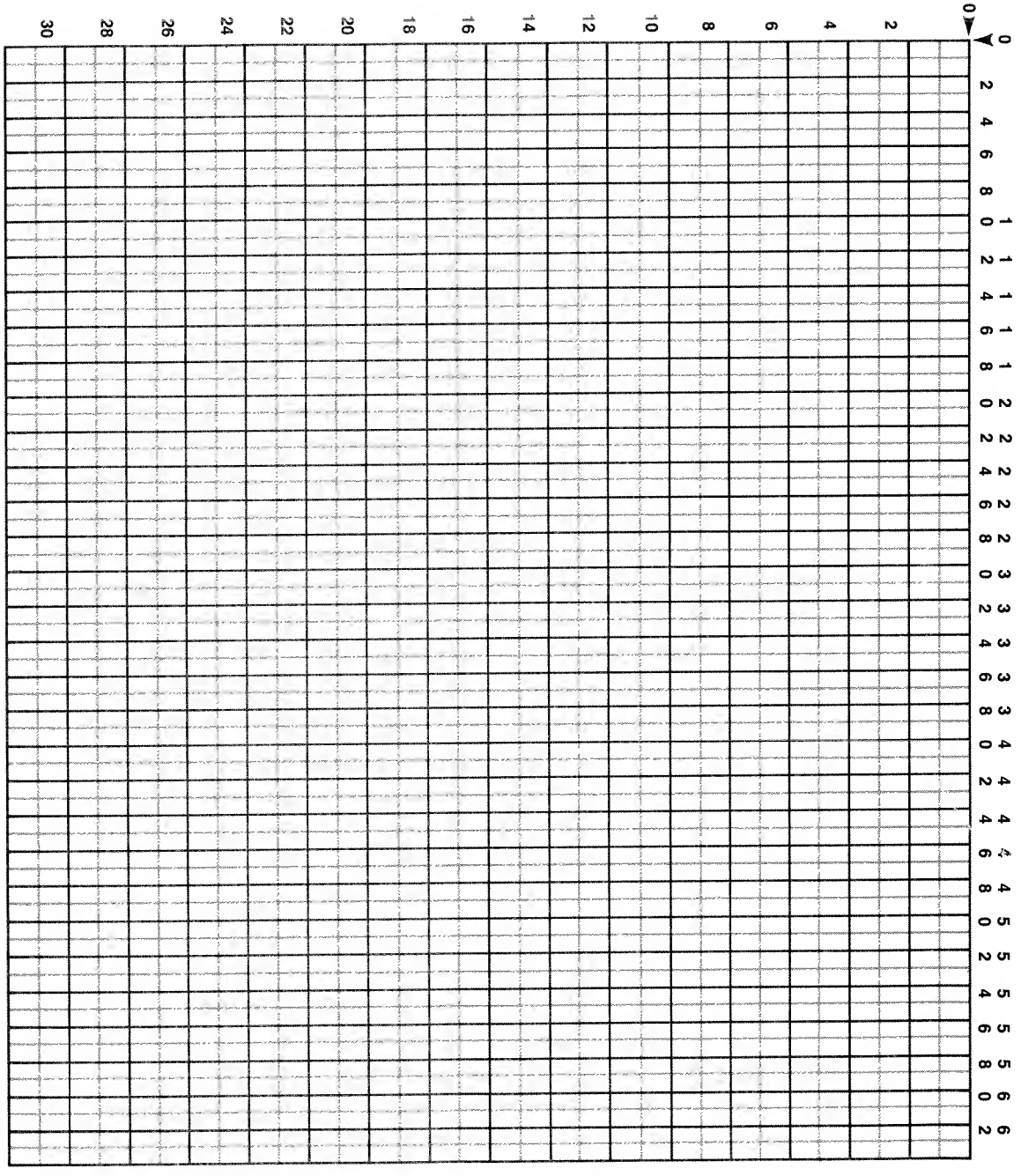
GRAPHICS SCREEN WORKSHEET (256 x 192)



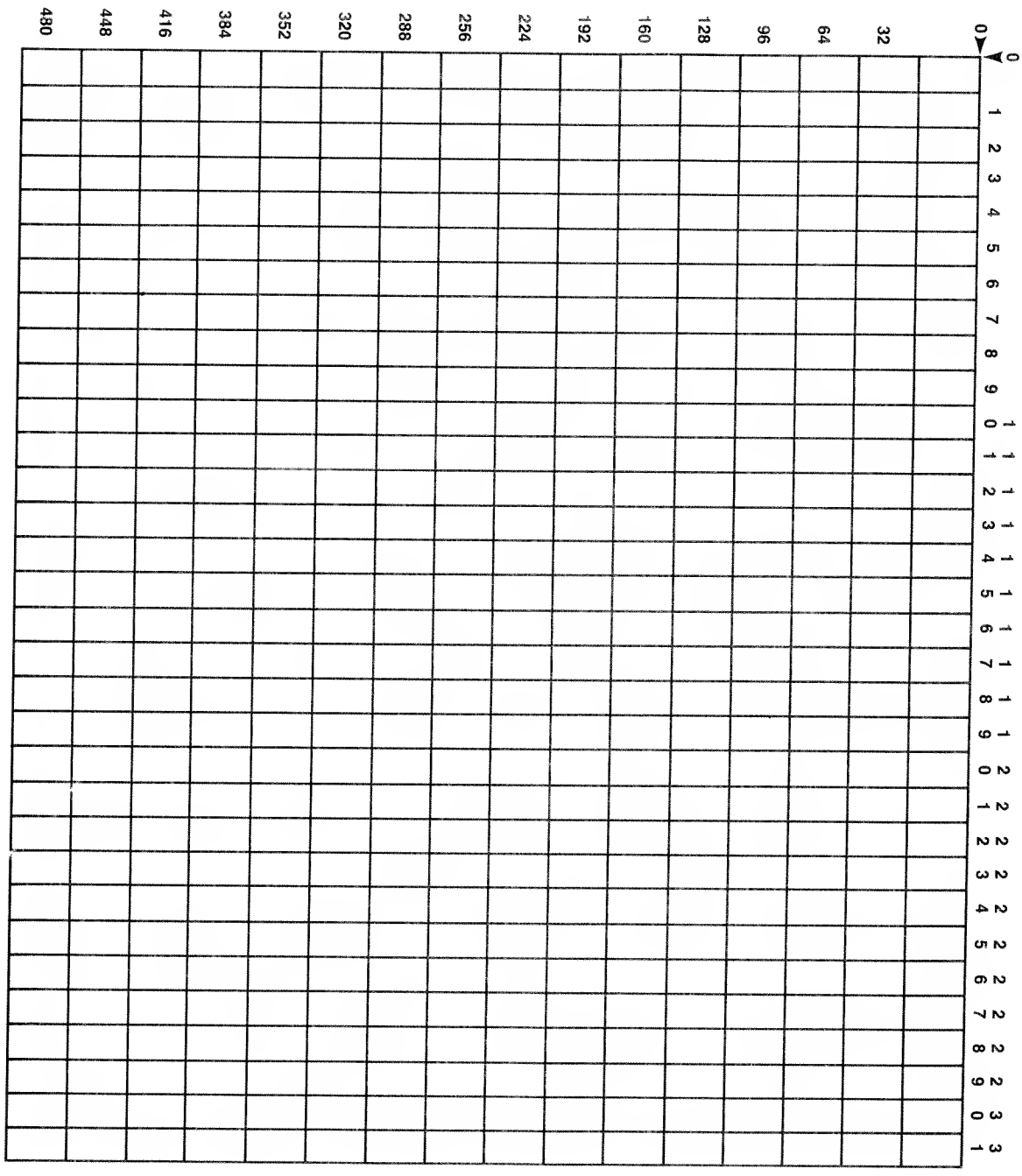
GRAPHICS SCREEN WORKSHEET (128 x 96)



SET/RESET WORKSHEET (64 x 32)



PRINT @ WORKSHEET (32 x 16)



Extended Color BASIC Colors

Here are the codes for the nine colors you can create on your computer:

Code	Color
0	Black
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

The color may vary in shade from these, depending on your TV. Color 0 (black) is actually an absence of color.

COLOR-SET

Mode	Color Set	Two-Color Combination	Four-Color Combination
4	0	Black/Green	—
	1	Black/Buff	—
3	0	—	Green/Yellow/Blue/Red
	1	—	Buff/Cyan/Magenta/Orange
2	0	Black/Green	—
	1	Black/Buff	—
1	0	—	Green/Yellow/Blue/Red
	1	—	Buff/Cyan/Magenta/Orange
0	0	Black/Green	—
	1	Black/Buff	—

MUSICAL NOTE/NUMBER

Number	Note
1	C
2	C#/D-
3	D
4	E-/D#
5	E/F-
6	F/E#
7	F#/G-
8	G
9	G#/A-
10	A
11	A#/B-
12	B

Note: PLAY does not recognize the notation B# or C−. Use the numbers 1 and 12 respectively or substitute C for B# and B for C−. A ?FC Error occurs if you try to use either of these notations.

Extended Color Basic Error Messages

Abbreviation	Explanation
/0	Division by Zero. It's impossible to divide by zero, even for computers.
AO	Attempt to Open a file that is already open. If you press RESET during cassette I/O, you'll get this message. Turn the computer off and on again.
BS	Bad Subscript. The subscripts in an array are out of range. For example, if you have A(12) in your program without a preceding DIM line that dimensions array A for 12 or more elements, you'll get this error. Use DIM to dimension the array.
CN	Can't Continue. If you use the CONT command and you're at the END of program or in other non-continue situations, you'll get this error.
DD	Attempt to Redimension an Array. You can dimenension an array only once. For example, you can't have DIM A(12) and DIM A(50) in the same program.
DN	Device Number Error. You may use only three device numbers with OPEN, CLOSE, PRINT, or INPUT—0, -1, or -2. If you use another number, you'll get this error.
DS	Direct Statement. The data file contains a direct statement. This error can be caused by attempting to CLOAD a data file.
FC	Illegal Function Call. This error occurs when you use a parameter (number or variable) with a BASIC word that is out of range. For example, PLAY":" causes this error.
FD	Bad File Data. This error occurs when you PRINT data to a file or INPUT data from the file, using the wrong type of variable for the corresponding data. For example, INPUT #-1,A, when the data in the file is a string, causes this error.
FM	Bad File Mode. This error occurs when you try to INPUT data from a file OPEN for OUTPUT(O), or PRINT data into a file OPEN for INPUT(I).

Abbreviation	Explanation
ID	Illegal Direct Statement. For example, you can use INPUT only as a line in the program, not as a command line.
IE	Input past End of file. Use EOF to check to see when you've reached the end of the file. When you have, CLOSE the file.
IO	Input/Output Error. This error is often caused by trying to input a program or data file from a bad tape.
LS	String too long. A string may contain only 255 characters.
NF	NEXT without FOR. NEXT is being used without a FOR statement. This error also occurs when you have the NEXT lines reversed in a nested loop.
NO	File Not Open. You can't input or output data to a file until you have OPENed it.
OD	Out of Data. A READ was executed with insufficient DATA for it to READ. A DATA statement may have been left out of the program.
OM	Out of Memory. All available memory has been used or reserved.
OS	Out of String Space. There is not enough space in memory to do your string operations. You may be able to CLEAR more space.
OV	Overflow. The number is too large for the computer to handle. (ABS(X)>1E38)
RG	RETURN without GOSUB. A RETURN line was encountered without a prior GOSUB.
SN	Syntax Error. This could result from a misspelled command, incorrect punctuation, open parentheses, or an illegal character. Retype the program line or command.
ST	String formula too complex. A string operation was too complex to handle. Break it into shorter steps.
TM	Type Mismatch. This occurs when you try to assign numeric data to a string variable (A$=3) or string data to a numeric variable (A="DATA").
UL	Undefined Line. The program contains a GOTO, GOSUB, or other branching line that asks the computer to go to a nonexisting line number.

A Formula in Hand Is Worth Two in the Book . . .

Quantity	Standard Formulas	BASIC Statement
Total Degrees of a Triangle	$180° = A + B + C$	`TTL = AA + AB + AC`
Solve for Area Given Side a, Angles B and C	$A = 180 - (B + C)$ $\text{Area} = \frac{a^2 \sin B \cdot \sin C}{2 \sin A}$	`AA = 180 - (AB + AC) [then convert AA, AB and AC to radians]` `AREA = SA^2*SIN(AB)*SIN(AC)/(2*SIN(AA))`
Given Sides a, b and c	$s = \frac{1}{2}(a+b+c)$	`S = (SA + SB + SC)/2`
Law of Sines	$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$ $\frac{a}{b} = \frac{\sin A}{\sin B}$ or $a = \frac{\sin A}{\sin B} \cdot b$	`AREA = SQR(S*(S - SA)*(S - SB)*(S - SC))` `SA = (SIN(AA)*SIN(AB))*SB`
Law of Cosines	$a^2 = b^2 + c^2 - 2bc \cdot \cos A$ or $a = \sqrt{b^2 + c^2 - 2bc \cdot \cos A}$	`SA = SQR(SB^2 - SC^2 - 2*SB*SC*COS(AA))`
Law of Tangents	$\frac{a-c}{a+c} = \frac{\tan \frac{1}{2}(A-C)}{\tan \frac{1}{2}(A+C)}$ or $\tan \frac{1}{2}(A-C) = \frac{a-c}{a+c} \cdot \tan \frac{1}{2}(A+C)$	`REM Y = TAN((AA - AC)/2)` `Y = (SA - SC)/(SA + SC)*TAN((AA + AC)/2)`

Given Three Sides, Solve for an Angle	$s = \frac{1}{2}(a+b+c)$
	$r = \sqrt{\frac{(s-a)(s-b)(s-c)}{s}}$
	$A = 2 \arctan\left(\frac{r}{s-a}\right)$
		`S = (SA + SB + SC)/2`
		`R = SQR((S - SA)*(S - SB)*(S - SC)/S)`
		`AA = 2*ATN(R/(S - SA))`

Quadratic Equations	$ax^2 + bx + c = 0$
	$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
		`REM A*X^2 + B*X + C = 0`
		`Z = B^2 - 4*A*C`
		`X1 = (-B + SQR(Z))/(2*A)  'IF Z>= 0`
		`X2 = (-B - SQR(Z))/(2*A)  'IF Z>= 0`

Algebraic Equations	$(a^x)^y = a^{xy}$
	$a^{-x} = \frac{1}{a^x}$
	$\log x^y = y \cdot \log x$
	$\log xy = \log x + \log y$
	$\log \frac{x}{y} = \log x - \log y$
		`Z = (A^X)^Y  or Z = A^(X*Y)`
		`Z = A^(-X)  or Z = 1/(A^X)`
		`Z = LOG(X^Y)  or Z = Y*LOG(X)`
		`Z = LOG(X*Y)  or Z = LOG(X) + LOG(Y)`
		`Z = LOG(X/Y)  or Z = LOG(X) - LOG(Y)`

Side a = *SA (side opposite Angle A)*
Side b = *SB (side adjacent to Angle A)*
Side c = *SC (hypotenuse)*
Angle A = *AA*
Angle B = *AB*
Angle C = *AC*

Function	Function Expressed in Terms of Extended Color BASIC Functions. x is in radians.
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(−X*X+1))
INVERSE COSINE	ARCCOS(X) = −ATN(X/SQR(−X*X+1))+1.5708
INVERSE SECANT	ARCSEC(X) = ATN(SQR(X*X −1))+(SGN(X)−1)*1.5708
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X*X−1))+(SGN(X)−1)*1.5708
INVERSE COTANGENT	ARCCOT(X) = −ATN(X)+1.5708
HYPERBOLIC SINE	SINH(x) = (EXP(X)−EXP(−X))/2
HYPOBOLIC COSINE	COSH(X) = (EXP(X)+EXP(−X))/2
HYPERBOLIC TANGENT	TANH(X) = −EXP(−X)/(EXP(X)+EXP(−X))*2+1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X)+EXP(−X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X)−EXP(−X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(−X)/(EXP(X)−EXP(−X))*2+1
INVERSE HYPERBOLIC SINE	ARGSINH(X) = LOG(X+SQR(X*X+1))
INVERSE HYPERBOLIC COSINE	ARGCOSH(X) = LOG(X+SQR(X*X−1))
INVERSE HYPERBOLIC TANGENT	ARGTANH(X) = LOG((1+X)/(1−X))/2
INVERSE HYPERBOLIC SECANT	ARGSECH(X) = LOG((SQR(−X*X+1)+1)/X)
INVERSE HYPERBOLIC COSECANT	ARGCSCH(X) = LOG((SGN(X)*SQR(X*X+1)+1)/X)
INVERSE HYPERBOLIC COTANGENT	ARGCOTH(X) = LOG((X+1)/(X−1))/2

Valid Input Ranges

Inverse Sine	-1<X<1
Inverse Cosine	-1<X<1
Inverse Secant	X<-or X>1
Inverse Cosecant	X<-1 or X>1
Inverse Hyper. Cosine	X>1
Inverse Hyper. Tangent	X*X<1
Inverse Hyper. Secant	0<X<1
Inverse Hyper. Cosecant	X<>0
Inverse Hyper. Cotangent	X*X>1

Certain special values are mathematically undefined, but our functions may provide invalid values:

TAN and SEC of 90 and 270 degrees
COT and SCS of 0 and 180 degrees

For example, TAN(1.5708) returns a value but TAN(90*.01745329) returns a DIVISION BY ZERO error. 90*.01745329 = 1.5708

Other values that are not available from these functions are:

ARCSIN(-1) = -PI/2
ARCSIN(1) = PI/2
ARCCOS(-1) = PI
ARCCOS(1) = 0
ARCSEC(-1) = -PI
ARCSEC(1 = 0
ARCCSC(-1) = -PI/2
ARCCSC(1) = PI/2

Please note that the above information may not be exhaustive.

Decimal Address	Contents	Hex Address
0-1023	System Use	0-3FF
255	Direct Page RAM	0FF
1023	Extended Page RAM	3FF
1024-1535	Text Screen Memory	400-5FF
	Graphic Screen Memory	
1536-3071	Page 1	600-BFF
3072-4607	Page 2	C00-11FF
4608-6143	Page 3	1200-17FF
6144-7679	Page 4	1800-1DFF
7680-9215	Page 5	1E00-23FF
9216-2559	Page 6	2400-9FF
2560-12287	Page 7	2A00-2FFF
12288-13823	Page 8	3000-35FF
	Program and Variable	
13824-16383	Storage	3600-3FFF
32768-40959	Extended Color BASIC	8000-9FFF
40960-49151	Color BASIC	A000-BFFF
49152-65279	Cartridge Memory	C000-FEFF
65280-65535	Input Output	FF00-FFFF

Color Computer Line Printer Variables

Variable	Hexadecimal Address	Decimal Address	Initial Hex	Value Dec
LPTBTD Baud				
MSB	0095	149	00	0
LSB	0096	150	57	87
LPTLND Line Delay				
MSB	0097	151	00	0
LSB	0098	152	01	1
LPTCFW Comma Field Width				
	0099	153	10	16
LPTLCF Last Comma Field				
	009A	154	70	112
LPTWID Line Printer Width				
	009B	155	84	132
LPTPOS				
	009C	156	00	00

Your computer's software uses the following initial conditions:

- The baud rate is 600
- The printer width is 132 columns
- The printer generates a busy output when not ready
- The printer automatically executes a carriage return at 132 columns.

The RS-232 Interface uses a four-pin DIN connector. A diagram of the pin out is shown in your introduction manual.

Pin 4 is the computer output to the printer. Pin 3 is ground. Pin 1 is not used for a printer. Pin 2 should be connected to the busy output (or status line) of the printer. If your printer does not provide a status indication, then this line must be connected to a positive voltage of greater than 3 volts. This tells the computer that the printer is ready at all times. In addition, the line delay variable should be set to the proper value.

The following list of alternate values for the line printer variables is provided as an aid in interfacing nonstandard printers.

Baud Rate	Decimal Value (msb,lsb)	Hexadecimal Value
120 baud	458 (1 and 202)	01CA
300 baud	180	00BE
600 baud	87	0057
1200 baud	41	0029
2400 baud	18	0012

Exit Conditions
None

CHROUT = [A002]
Outputs a Character to Device

CHROUT outputs a character to the device specified by the contents of 6F (DEVNUM).
DEVNUM = –2 (printer)
DEVNUM = 0 (screen)

Entry Conditions
On entry, the character to be output is in A.

Exit Conditions
All registers except CC are preserved.

CSRDON = [A004]
Starts Cassette

CSRDON starts the cassette and gets into bit sync for reading.

Entry Conditions
None

Exit Conditions
FIRQ and IRQ are masked. U and Y are preserved. All others are modified.

GIVABF = [B4F4]
Passes parameter to BASIC

Entry Conditions
D = parameter

Exit Conditions
USR variable = parameter

INTCNV = [B3ED]
Passes parameter from BASIC

Entry Conditions
USR argument = parameter

Exit Conditions
D = parameter

JOYIN = [A00A]
Samples Joystick Pots

JOYIN samples all four joystick pots and stores their values in POTVAL through POTVAL + 3.

Left Joystick	
Up/Down	15A
Right/Left	15B
Right Joystick	
Up/Down	15C
Right/Left	15D

For Up/Down, the minimum value = UP.
For Right/Left, the minimum value = LEFT.

Entry Conditions
None

ROM Routines

The Color BASIC ROM contains many subroutines that can be called by a machine-language program. Each subroutine will be described in the following format:

NAME — *Entry address*
Operation Performed
Entry Condition
Exit Condition

> ***Note:*** *The subroutine* ***NAME*** *is only for reference. It is not recognized by the Color Computer. The* ***entry address*** *is given in hexadecimal form; you must use an indirect jump to this address.* ***Entry*** *and* ***Exit Conditions*** *are given for machine-language programs.*

BLKIN = [A006]
Reads a Block from Cassette

Entry Conditions
Cassette must be on and in bit sync (see CSRDON). CBUFAD contains the buffer address.

Exit Conditions
BLKTYP, which is located at 7C, contains the block type:

0 = File Header
1 = Data
FF = End of File

BLKLEN, located at 7D, contains the number of data bytes in the block (0-255).
Z* = 1,A = CSRERR = 0 (if no errors).
Z = 0, A = CSRERR = 1 (if a checksum error occurs).
Z = 0, A = CSRERR = 2 (if a memory error occurs).

> ***Note:*** *CSRERR = 81*

Unless a memory error occurs, X = CBUFAD + BLKEN. If a memory error occurs, X points to beyond the bad address. Interrupts are masked. U and Y are preserved, all other modified.

**Z is a flag in the Condition Code (CC) register.*

BLKOUT = [A008]
Writes a Block to Cassette

Entry Conditions
The tape should be up to speed and a leader of hex 55s should have been written if this is the first block to be written after a motor-on.
CBUFAD, located at 7E, contains the buffer address.
BLKTYP, located at 7C, contains the block type.
BLKLEN, located at 7D, contains the number of data bytes.

Exit Conditions
Interrupts are masked.
X = CBUFAD + BLKLEN.
All registers are modified.

WRTLDR = [A00C]
Turns the Cassette On and Writes a Leader

Entry Conditions
None

Exit Conditions
None

CHROUT = [A002]
Outputs a Character to Device

CHROUT outputs a character to the device specified by the contents of 6F (DEVNUM).
DEVNUM = –2 (printer)
DEVNUM = 0 (screen)

Entry Conditions
On entry, the character to be output is in A.

Exit Conditions
All registers except CC are preserved.

CSRDON = [A004]
Starts Cassette

CSRDON starts the cassette and gets into bit sync for reading.

Entry Conditions
None

Exit Conditions
FIRQ and IRO are masked. U and Y are preserved. All others are modified.

GIVABF = [B4F4]
Passes parameter to BASIC

Entry Conditions
D = parameter

Exit Conditions
USR variable = parameter

INTCNV = [B3ED]
Passes parameter from BASIC

Entry Conditions
USR argument = parameter

Exit Conditions
D = parameter

JOYIN = [A00A]
Samples Joystick Pots

JOYIN samples all four joystick pots and stores their values in POTVAL through POTVAL + 3.

Left Joystick	
Up/Down	15A
Right/Left	15B
Right Joystick	
Up/Down	15C
Right/Left	15D

For Up/Down, the minimum value = UP.
For Right/Left, the minimum value = LEFT.

Entry Conditions
None

Exit Conditions
Y is preserved. All others are modified.

POLCAT = [A000]
Polls Keyboard for a Character

Entry Conditions
None

Exit Conditions
Z = 1, A = 0 (if no key seen).
Z = 0, A = key code, (if key is seen).
B and X are preserved. All others are modified.

BASIC SUMMARY

STATEMENTS

BASIC statements are commands that tell your computer to do some action, such as drawing a circle on the screen. Use BASIC statements as lines in your program.

AUDIO Connects or disconnects cassette output to TV speaker.

CIRCLE *(x,y),r,c,hw,start,end* Draws a circle with center at point (*x*,*y*), radius *r*, specified color *c*, height/width ratio (*hw*) of 0-4. Circle can start and end at specified point (0-1).

CLEAR *n,h* Reserves *n* bytes of string storage space. Erases variables. *h* specifies highest BASIC address.

CLOAD Loads specified program file from cassette. If *filename* is not specified, first file encountered is loaded. *Filename* can be a maximum of 8 characters.

CLOADM Loads machine-language program cassette. You may specify an offset address to add to the loading address.

CLOSE*#DEV* Closes access to specified file. If you do not specify *device*, all open files are closed.

CLS *c* Clears display to specified color *c*. If you do not specify color, green is used.

COLOR *(foreground,background)* Sets foreground and background color.

CONT Continues program execution after you have pressed **BREAK** or used the STOP statement.

CSAVE Saves program on cassette (program name can be 8 characters or fewer). If you specify A, program is saved in ASCII format.

CSAVEM *name, start, end, transfer* Saves a machine-language file on cassette.

DATE Stores data in your program. Use READ to assign data to variables.

DEF FN Defines numeric function.

DEFUSR *n* Defines entry point for USR function n.n = 0-9.

DEL Deletes program lines.

DIM Dimensions one or more arrays.

DRAW Draws a line beginning at specified starting point of specified length of specified color. Also draws to scale, draws blank lines, draws nonupdated lines, and executes substrings. If you do not specify starting point, last DRAW position or (128,96) is used.

EDIT Lets you edit a program line.

END Ends program.

EXEC (*address*) Transfers control to machine-language programs at specified address. If you omit address, control is transferred to address set in last CLOADM.

FOR . . . TO STEP/NEXT Creates a loop in program that the computer must repeat from the first number to the last number you specify. Use STEP to specify how much to increment the number each time through the loop. If you omit STEP, the computer uses 1.

GET *(start)-(end),destination,G* Reads the graphic contents of a rectangle into an array for future use by PUT.

GOSUB Calls a subroutine beginning at specified line number.

GOTO Jumps to specified line number.

IF *test* THEN . . . *action 1* ELSE, *action 2* Performs a test. If it is true, the computer executes *action1*. If false, the computer executes *action 2*.

INPUT Causes the computer to stop and await input from the keyboard.

INPUT#-1 Input data from cassette.

INSTR *(position, search, target)* Searches for the first occurrence of target string in search string beginning at position. Returns the position at which the match is found.

LET Assigns value to variable (optional).

LIST Lists (displays) specified line(s) or entire program on screen.

LLIST Lists specified program line(s) or entire program to printer.

LINE *(x1,y1)-(x2,y2)*, PSET or PRESET,BF Draws a line from *(x1,y1)* to *(x2,y2)*. If you omit *(x1,y1)*, the last end point or (128,96) is used. PSET selects foreground color, and PRESET selects background color. B draws a box with *(x1,y1)* and *(x2,y2)* as the opposing corners. BF fills in the box with foreground color.

LINE INPUT Inputs line form keyboard.

MID$ *(oldstr, position, length)* Replaces a portion of *oldstr* with another string.

MOTOR Turns cassette ON or OFF.

NEW Erases everything in memory.

ON . . . GOSUB Multiway branch to call specified subroutines.

ON . . . GOTO Multiway branch to specified lines.

OPEN *m,#dev,f* Opens specified file (*f*) for data transmission (*m*) to specified device (*dev*). *m* may be I (Input) or O (Output). *dev* may be #0 (screen or keyboard), #-1 (cassette), or #-2 (printer).

PAINT *(x,y),c,b* Paints graphic screen starting at point *(x,y)* with specified color *(c)* and stopping at border *(b)* of specified color.

PCLEAR *n* Reserves *n* number of 1.5 K graphics memory pages.

PCLS *c* Clears screen with specified color *(c)*. If you omit color code, current background color is used.

PCOPY Copy graphics from source page to destination page.

PLAY Plays music of specified note (A-G or 1-12), octave (O), volume (V), note-length (L), tempo(T), pause (P), and allows execution of substrings. Also sharps (# or +) and flats (-).

PMODE *mode, start-page* Selects resolution and first memory page.

POKE *(location, value)* Puts value (0-255) into specified memory location.

PRESET Resets a point to background color.

PRINT Prints (displays) specified message or number on TV screen.

PRINT #-1 Writes data to cassette.

PRINT #-2 Prints an item or list of items on the printer.

PRINT TAB Moves the cursor to specified column position.

PRINT USING Prints numbers in specified format.

PRINT @ *scr pos* Prints specified message at specified text screen position.

PSET *(x,y,c)* Sets a specified point *(x,y)* to specified color *(c)*. If you omit c, foreground is used.

PUT *(start)-(end), source, action* Stores graphics from source onto start/end rectangle on the screen. (Array rectangle size must match GET rectangle size.)

READ Reads the next item in DATA line and assigns it to specified variable.

REM Lets you insert comment in program line. The computer ignores everything after REM.

RENUM *newline, startline, increment* Lets you renumber program lines.

RESET *(x,y)* Resets a point.

RESTORE Sets the computer's pointer back to first item on the first DATA line.

RETURN Returns the computer from subroutine to the BASIC word following GOSUB.

RUN Executes a program.

SCREEN *screen-type, color-set* Selects either graphics (1) or text (0) screen and color-set (0 or 1).

SET *(x,y,c)* Sets a dot at specified text screen position to specified color.

SKIPF Skips to next program on cassette tape or to end of specified program.

SOUND *tone, duration* Sounds specified tone for specified duration.

STOP Stops execution of a program.

TROFF Turns off program tracer.

TRON Turns on program tracer.

FUNCTIONS

BASIC functions are built-in subroutines that perform some kind of computation on data, such as computing the square root of a number. Use BASIC functions as data within your program lines.

ABS ***(numeric)*** Computes absolute value.

ASC ***(str)*** Returns ASCII code of first character of specified string.

ATN ***(numeric)*** Returns arctangent in radians.

CHR$ ***(code)*** Returns character for ASCII, control, or graphics code.

COS ***(numeric)*** Returns cosine of an angle given in radians.

EOF ***(dev)*** Returns FALSE = 0 if there is more data; TRUE = –1 if end of file has been read.

EXP ***(numeric)*** Returns natural exponential of number (e *number*).

HEX$ ***(numeric)*** Computes hexadecimal value. PRINT HEX$ (30)

INKEY$ Checks the keyboard and returns the key being pressed (if any).

INT ***(numeric)*** Converts a number to an integer.

JOYSTK ***(j)*** Returns the horizontal or vertical coordinate (j) of the left or right joystick:
0 = horizontal, left joystick
1 = vertical, left joystick
2 = horizontal, right joystick
3 = vertical, right joystick

LEN ***(str)*** Returns the length of a string.

LOG ***(numeric)*** Returns natural logarithm.

MEM Finds the amount of free memory.

MID$ ***(str,pos,length)*** Returns a substring of another string starting at *pos*. If you omit *length*, the entire string right of position is returned.

PEEK ***(mem loc)*** Returns the contents of specified memory location.

POINT ***(x,y)*** Tests whether specified graphics cell is on or off. *x* (horizontal) = 0–63; *y* (vertical) = 0–31. The value returned is –1 if the cell is in a text character mode; 0 if it is off, or the color code if it is on. See CLS for color codes.

POS ***(dev)*** Returns current print position.

PPOINT ***(x,y)*** Tests whether specified graphics cell is on or off and returns color code of specified cell.

RIGHT$ ***(str,length)*** Returns right portion of string.

RND ***(n)*** Generates a "random" number between 1 and *n* if $n > 1$, or between 0 and 1 if $n = 0$.

SGN ***(numeric)*** Returns sign of specified numeric expression: –1 = negative; 0 = 0; 1 = positive.

SIN ***(numeric)*** Returns sine of angle given in radians.

STRING$ ***(length, code, or string)*** Returns a string of characters (of specified length) specified by ASCII code or by the first character of the string.

STR$ ***(numeric)*** Converts a numeric expression to a string.

SQR ***(numeric)*** Returns the square root of a number.

TAN ***(numeric)*** Returns tangent of angle given in radians.

TIMER Returns contents or lets you set timer (0-65535).

USR*n* ***(numeric)*** Calls your machine-language subroutine.

VAL ***(str)*** Converts a string to a number.

VARPTR ***(var)*** Returns addresses of pointer to the specified variable.

OPERATORS

BASIC operators perform some kind of operation on data, such as adding two numbers.

↑	Exponentiation
–,+	Unary negative, positive
*,/	Multiplication, division
+,–	Addition and concatenation, subtraction
<,>,=,<=,>=,<>	Relational tests
NOT	
AND	
OR	

INDEX

RADIO SHACK A DIVISION OF TANDY CORPORATION

U.S.A.
FORT WORTH, TEXAS 76102

CANADA
BARRIE, ONTARIO, L4M4W5

TANDY CORPORATION

AUSTRALIA
280-316 VICTORIA ROAD
RYDAMERE, N.S.W. 2116

BELGIUM
PARC INDUSTRIEL NANINNE
5140 NANINNE

UNITED KINGDOM
BILSTON ROAD, WEDNESBURY
WEST MIDLANDS WS10 7JN